UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA DEI SISTEMI,

DEL TERRITORIO E DELLE COSTRUZIONI

RELAZIONE PER IL CONSEGUIMENTO DELLA

LAUREA MAGISTRALE IN INGEGNERIA GESTIONALE

Sviluppo di un Applicativo per i Contact Center Inbound tramite Approccio Agile

---

RELATORI

Prof. Ing. Roberto Mirandola

Dipartimento di Ingegneria Civile e Industriale

Ing. Ilaria Campana

Consorzio QUINN

IL CANDIDATO

Francesca Trossarelli

francitros@fastwebnet.it

Sessione di Laurea del 25/11/2015
Anno Accademico 2014/2015
Consultazione NON consentita

Sviluppo di un Applicativo per i Contact Center Inboud tramite Approccio Agile

Francesca Trossarelli

Sommario

La velocità con cui si evolvono e diventano obsolete le tecnologie e la centralità dei clienti e degli utilizzatori finali per le aziende, fanno sì che vengano richiesti sforzi sempre maggiori per aumentare la rapidità nello sviluppo dei prodotti e coinvolgere in questa fase le figure a cui il prodotto è indirizzato. Tutto questo, seppur variabile a seconda della realtà industriale di riferimento, è un punto fondamentale nel mondo dello sviluppo di prodotti software. Nella seguente tesi vengono confrontati l'utilizzo di metodologie tradizionali nello sviluppo degli applicativi software, quali il metodo Waterfall, con approcci più recenti, che si contraddistinguono per la loro snellezza e velocità nell'ottenere risultati: le metodologie Agile. Tale confronto verrà supportato dall'osservazione e partecipazione ad un caso reale di utilizzo dell'approccio Agile, avvenuto durante lo sviluppo di un applicativo per i Call Center Inbound di una nota azienda di telecomunicazioni. Questo lavoro ha il fine di capire se effettivamente questi metodi innovativi di gestione e organizzazione del lavoro sono superiori rispetto agli approcci tradizionali e in quali contesti potrebbero essere ancora preferibili i così detti metodi pesanti.

Abstract

The speed at which technologies evolve and become obsolete, together with the importance companies assign to their clients and end-users, make it imperative that greater and greater efforts be spent to increase the development of products while also involving in this phase the people to which the product is aimed. All of this, though extremely variable depending on the specific situation, remains a significant point in the development of software products. In the following thesis the utilization of more traditional methods in the development of software such as the Waterfall method are confronted with those more recent approaches which tend to be more immediate and able to obtain quicker results: the Agile methodologies. This study is based on the observation and the participation of a true case utilizing the Agile approach during the development of software for Inbound Contact Centers of a well-known telephone company. The work is aimed at better understanding if these innovative methods are actually more efficient compared to the more traditional approaches and in which contexts on the other hand these could still be preferable.

Indice

Indice delle figure

Indice delle tabelle

1 | INTRODUZIONE

In un mondo sempre più tecnologico, lo sviluppo di nuovi software acquisisce un'importanza maggiore e a causa dell'avvento di Internet e della turbolenza dei mercati diventa imperativo farlo in modo veloce e preciso, in modo da precedere la rapida obsolescenza dei prodotti. La crescente importanza della figura del cliente, che con le sue esigenze e le sue aspettative diventa il focus imprescindibile per qualsiasi impresa che voglia sopravvivere nel contesto economico, fa sì che anche nell'ambito dello sviluppo dei software, siano i suoi requisiti a guidare la realizzazione del prodotto. Per questi motivi è stato necessario spostarsi da metodologie pesanti, che richiedono tempi lunghi e un minor coinvolgimento dell'utente finale, a metodologie snelle, che prevedono rilasci in tempi rapidi e il focus sulla collaborazione con il cliente/utente al fine di recepire le sue richieste in modo puntuale. Questo insieme di metodi, che stanno prendendo sempre più piede negli ultimi quindici anni, prende il nome di Metodologia Agile.

Nonostante la tendenza di spostarsi sempre più verso l'Agile, si richiede un'analisi più approfondita per capire se questa è la scelta più efficace per tutti i contesti e per vedere quali sono le implicazioni che l'usare una metodologia piuttosto di un'altra, nella medesima situazione, comporta.

In tale ambito nasce questo lavoro di tesi, che prevede un excursus dalle metodologie tradizionali a quelle Agile, fino ad approdare ad un caso reale di utilizzo di quest'ultime. Il caso si riconduce all'esperienza maturata durante lo stage presso la società di consulenza Accenture S.p.A., la quale ha partecipato ad un progetto di sviluppo ed implementazione di un applicativo sui dispositivi dei Contact Center inbound di una delle più grandi società di telecomunicazione italiane. All'interno di questa parte si cercherà di individuare l'ambito in cui nasce un progetto di questo tipo e descrivere i principali step di cui si compone, riconducendo il metodo usato alle descrizioni teoriche viste in precedenza. In particolare il focus verrà fatto sulla parte di

raccolta dei requisiti, della quale mi sono occupata in prima persona, e sullo sviluppo del prototipo da mostrare al cliente/utente finale. Infine si cercherà di capire cosa avrebbe comportato l'adozione di una metodologia diversa e quali sono gli ambiti in cui è preferibile l'utilizzo di approcci tradizionali rispetto a quelli Agile.

2 | METODOLOGIE TRADIZIONALI

In questo capitolo verrà presentata una descrizione delle metodologie tradizionali, anche dette "prescrittive" o "convenzionali", utilizzate per gestire specialmente i progetti riguardanti lo sviluppo dei software. I processi che regolano tale sviluppo sono infatti alla base dello sviluppo tecnologico degli applicativi e ne permettono un avanzamento razionale e coerente con le tempistiche desiderate. Grazie a questi processi infatti viene definita un'architettura in cui sono presenti delle KPAs (*key process areas*), le quali permettono il controllo gestionale del processo e consentono l'applicabilità dei metodi lavorativi, la produzione di documenti, *form*, *report* e dati. Inoltre tali KPAs assicurano la qualità e la gestione del cambiamento.

Tipicamente un processo di sviluppo software è organizzato come rappresentato in Figura 2.1:

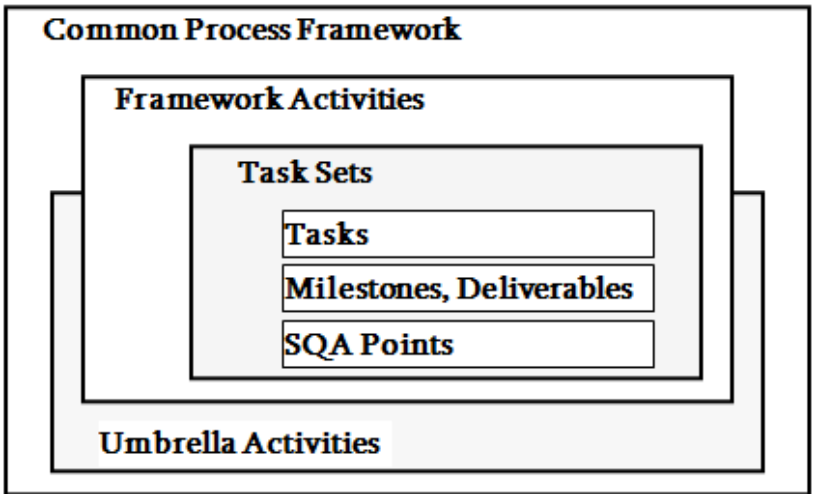

Figura 2.1: *Processo di sviluppo dei software*

Inizialmente vengono definite un certo numero di *Framework Activity*, indipendentemente dalle dimensioni e dalla complessità del progetto. Per adattare tali attività alle caratteristiche del progetto e alle richieste del *team* di progetto, all'interno

di queste viene a sua volta definito un certo numero di *task set* che comprendono compiti, *deliverable* e *check point relativi alla qualità*, variabili da progetto a progetto. Le *Framework Activity* solitamente definite per lo sviluppo di un qualsiasi applicativo sono le seguenti:

- Definizione del progetto grazie alla comunicazione col cliente e alla definizione dei requisiti,
- Pianificazione degli obiettivi, delle risorse, delle tempistiche e valutazione dei rischi gestionali e tecnici,
- Ingegnerizzazione e costruzione tramite codificazione e testing,
- Rilascio del software nel suo ambiente e supporto al cliente.

Come è possibile notare dalla Figura 2.1, trasversalmente alle *Framework Activity* si hanno le così dette *Umbrella Activity*, che sono indipendenti da qualsiasi delle attività sopra citate. Tra queste si hanno:

- Gestione del progetto,
- Quality Assurance,
- Definizione, utilizzo e analisi delle metriche,
- Gestione del rischio,
- Preparazione e stesura dei documenti.

Entrando nel vivo della trattazione, a livello pratico, nell'ambito dell'applicabilità industriale, a ciascuno di questi processi deve essere associata una strategia a seconda della natura del progetto e in base alle tempistiche, ai controlli, ai metodi e agli strumenti utilizzati, in modo da risolvere i problemi effettivi che possono presentarsi. Tali strategie non sono nient'altro che le metodologie, tradizionali o Agile, di cui verrà esposta, in questo capitolo e nel seguente, una trattazione delle principali.

2.1 Modello a Cascata

Il modello a cascata (*waterfall model*) viene chiamato anche *classic life cycle* o *linear sequential model* e fu citato per la prima volta il 29 Giugno del 1956 da Herber D. Benington per descrivere i metodi di programmazione avanzata. Tuttavia la prima descrizione formale viene attribuita a Winston Walker Royce nel 1970, anche se egli non usò mai il termine "*waterfall*", che venne usato per la prima volta da Bell and Thayer in un articolo del 1976. Grazie alla teorizzazione di questo modello il processo di sviluppo del software venne inteso per la prima volta come un vero e proprio processo industriale, assimilabile a quello delle industrie manifatturiere e di costruzione.

Nel modello a cascata, il progetto è organizzato in una sequenza di fasi documentate, ciascuna delle quali produce un *output* che costituisce l'*input* per la fase successiva. La suddivisione del processo in tali fasi può essere generalizzata in quella rappresentata in Figura 2.2:

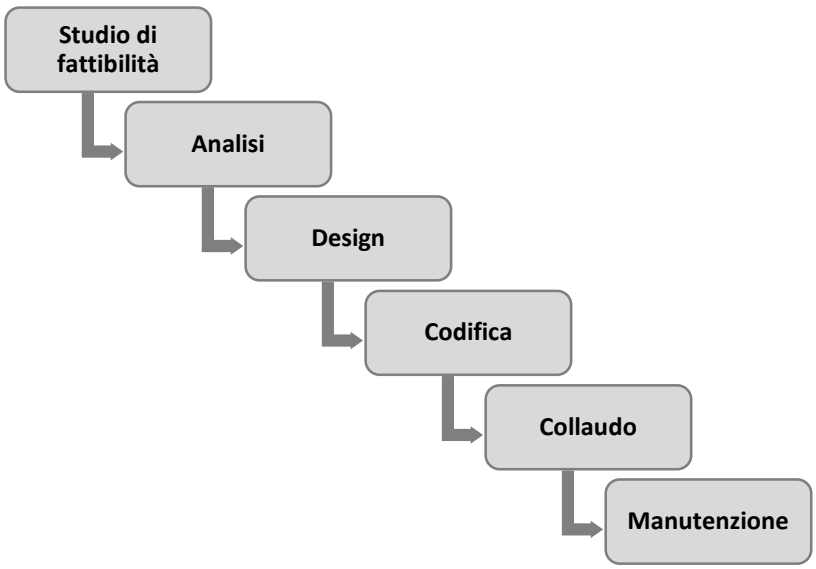

Figura 2.2: *Modello a cascata*

Studio di fattibilità: Questa fase consente di decidere se intraprendere o meno lo sviluppo del sistema, sia da un punto di vista tecnico che economico. Tale fase si conclude con la decisione (o il diniego), da parte del *Management,* di intraprendere il progetto

Analisi dei requisiti: Durante questa fase si ha un rilevante scambio informativo tra il team di progetto ed il committente e vengono raccolti i requisiti riguardanti sia il software che il sistema stesso. Gli analisti del team devono essere in grado di recepire sia le informazioni relative al dominio del software, sia i requisiti funzionali/comportamentali che le interfacce. Tali requisiti devono essere trascritti e raccolti in un apposito documento (*documento dei requisiti di prodotto*) che servirà da input per la fase successiva.

Design: In questa fase vengono tradotti i requisiti raccolti in una rappresentazione del software, sottoponibile ad un controllo qualitativo, prima ancora di essere effettivamente codificata. Tale fase serve a definire la struttura dei dati, l'architettura del sistema, la sua suddivisione in moduli e le interfacce tra questi.

Codifica: Nel corso di questa fase vengono effettivamente costruiti i moduli del sistema con un linguaggio di programmazione. Più la fase precedete viene svolta nel dettaglio, maggiormente questa fase può essere svolta in maniera meccanica.

Collaudo: Questa fase viene svolta conducendo test su 3 diverse aree. Un test funzionale eseguito dai programmatori per verificare la corretta installazione dei moduli, un test di integrazione eseguito dagli analisti funzionali per verificare la correttezza complessiva del Sistema e un test di accettazione svolto dal committente per verificare la rispondenza ai requisiti del Sistema.

Manutenzione: In seguito all'implementazione del software nel suo ambiente di destinazione, verranno svolte nel tempo operazioni di supporto, mantenimento estensione e miglioramento, a seguito di inevitabili cambiamenti del software stesso e delle condizioni a contorno.

Come detto in precedenza quella appena descritta è solo una generalizzazione del modello a cascata. Tale modello infatti può essere ridefinito con varianti specifiche a seconda dell'organizzazione e delle necessità che si manifestano. Tipicamente viene utilizzato per piccoli progetti, poiché, in questi casi, i requisiti iniziali sono chiaramente identificati.

2.1.1 Pro e Contro del Modello a Cascata

Dopo una breve descrizione del modello è interessante percepire e comprendere i vantaggi e gli svantaggi che la sua adozione comporta. Per quanto concerne le motivazioni a sostegno del suo utilizzo, esse possono essere riassunte in questo elenco:

- Solitamente nell'utilizzo di questa metodologia viene speso il 20-40% del tempo nelle fasi iniziali, il 30-40% nella fase di codifica e la restante percentuale di tempo per le ultime fasi. Grazie alla quantità di tempo spesa nelle fasi iniziali del progetto, possono essere risparmiati ingenti costi nelle fasi successive. Tale modello infatti si basa sul concetto di *Big Design Up Front* ovvero su una "grande progettazione a monte": infatti qualora uno stesso problema venisse riscontrato in una delle fasi iniziali piuttosto che in una successiva, esso sarebbe più facilmente eliminabile e si avrebbe un risparmio in termini monetari e di tempistiche. Su questo argomento Steve Mc Connell, ingegnere del software, afferma "*rimediare a un difetto nei requisiti che rimane nascosto sino alla fase di implementazione o di manutenzione del sistema, verrà a costare da 50 a 200 volte tanto, se comparato a ciò che si sarebbe speso se il difetto fosse stato rilevato durante la raccolta dei requisiti*".
- Applicando questo approccio ogni fase deve essere pienamente completata prima che si possa procedere alla successiva. Questo dovrebbe permettere di evitare situazioni in cui, accorgendosi di errori o problemi in fasi successive, si debba tornare indietro per risolverli, generando costi ingenti e onerose perdite di tempo.

- All’interno di questo modello viene data una particolare rilevanza alla documentazione, questo permette di trasmettere facilmente il know-how acquisito a nuovi membri del team o addirittura ad interi nuovi team.

- Grazie ad un approccio fortemente strutturato questo modello è di facile comprensione e consente di migrare da una fase alla successiva diminuendo errori e incomprensioni, aumentando teoricamente la disciplina all’interno del team e il rispetto delle tempistiche. Inoltre la presenza di traguardi ben identificabili permette di infondere una sensazione di realizzazione, ogni volta che uno di questi viene raggiunto.

Per quanto riguarda i 3 maggiori svantaggi del modello a cascata essi vengono evidenziati qui di seguito:

- Raramente i progetti reali seguono il flusso lineare del modello, perciò dei cambiamenti possono destabilizzare il team del progetto. Il modello perciò appare essere troppo rigido per qualcosa di complesso come lo sviluppo dei software.
- La necessità, imposta dal modello, di raccogliere tutti i requisiti nelle fasi iniziali risulta spesso controproducente. Questo è dovuto al fatto che spesso i committenti sono inconsapevoli di quali siano le necessità che dovranno essere soddisfatte da un software che ancora non conoscono. Questo problema, oltre al fatto di non recepire completamente tutte le richieste in fase iniziale, provoca l’aggiunta o la modifica di requisiti a metà del progetto, costringendo così a ripercorrere alcune fasi, col conseguente incremento di tempi e costi. In alcuni casi addirittura non sono possibili cambiamenti in corso del progetto, compromettendone così la riuscita ancora prima della fine.
- Le tempistiche legate a questo approccio sono molto lunghe e il cliente deve aspettare le fasi finali per vedere una versione funzionante del programma. L’impossibilità di testare il programma se non in fase finale può provocare

enormi danni qual ora ci siano dei problemi che non sono stati rilevati in precedenza.

Un'altra critica al modello, sostenuta da M. Bradac durante i suoi studi su progetti reali, è quella legata ai così detti *"blocking states".* Secondo questo studio durante le fasi iniziali e finali dei modelli lineari si verificano delle situazioni di blocco dovute al fatto che alcuni membri del progetto devono aspettare che altre persone concludano le loro attività per proseguire il lavoro. Queste fasi di inoperatività, nei casi più critici, eccedono persino i momenti di reale lavoro, causando così uno spreco (disomogeneo impiego) di risorse all'interno del progetto.

Altre critiche minori si aggiungono a quelle già viste e proprio a causa di tutte queste problematiche sono state sviluppate nel tempo delle variazioni più dinamiche del modello a cascata. Infine la necessità di flessibilità, dovuta all'alto ritmo di sviluppo e al flusso costante di cambiamenti a specifiche e funzionalità, ha fatto sì che il modello a cascata, con la sua rigidità, diventasse obsoleto e che si aprisse spazio per nuovi approcci.

2.2 Modello Prototipale

Nel caso in cui un cliente riesca ad esprimere solo obiettivi generici senza scendere nel dettaglio nell'individuazione dei requisiti del software o nel caso in cui il team di progetto sia insicuro su un algoritmo o sull'interazione tra l'applicativo e l'utente finale si ricorre all'utilizzo del *Modello Prototipale.* Ovviamente molti altri sono i casi in cui è conveniente adottare un approccio di questo tipo, il quale si basa sullo sviluppo di un prototipo, altrimenti chiamato *mock-up*. Come mostrato in Figura 2.3 questo modello inizia con una fase di comunicazione tra il team di progetto e il cliente, in modo tale da raccogliere tutti requisiti possibili e definire le aree per cui è necessaria una

delineazione più precisa. Successivamente viene prodotto un *"quick design"*, il quale si concentra principalmente sulle funzioni visibili dal cliente. A seguito di questa fase viene prodotto il primo prototipo che verrà valutato direttamente dal cliente/utente finale, in modo da ridefinire i requisiti necessari a svilupparlo.
Il prototipo non solo permette al cliente di soddisfare a pieno i suoi bisogni, ma consente anche al team di progetto di capire al meglio cosa debba essere fatto.

Figura 2.3: *Modello Prototipale*

Esistono due tipi di approcci che possono essere utilizzati nello sviluppare un prototipo:

1) *The Throw-Away Approach:* In questo caso il prototipo viene buttato via dopo avere adempiuto ai suoi doveri nelle fasi iniziali. Questo approccio viene utilizzato quando deve essere dimostrata la fattibilità di un nuovo concetto, al fine di raccogliere i fondi necessari per far partire il progetto. L'utilizzo di un prototipo che verrà gettato nelle fasi successive permette inoltre di adoperare linguaggi e strumenti speciali, che molto probabilmente non potranno essere usati successivamente nell'ambiente operativo.

   Per quanto riguarda gli svantaggi maggiori riconducibili a questo approccio essi si riconducono al fatto di spendere energie nell'implementazione di un codice che non verrà utilizzato nell'implementazione finale e nel ritenere inutile la documentazione riguardante questa fase. Spesso infatti si è tentati di

abbreviare o addirittura eliminare la documentazione, rischiando così di vanificare gli sforzi e disperdere le conoscenze apprese durante questa fase.

2) *The Evolutionary Approach:* In questo caso vengono prodotti una serie di prototipi fino a giungere al prodotto finale. Solitamente vengono mantenuti il modello concettuale e il design del prototipo, ma vengono utilizzate delle tecniche e degli strumenti appositi per migliorare il codice e definire i dettagli del prototipo prima di trasformarlo nell'applicativo finito, pronto per essere utilizzato.

2.2.1 Pro e Contro del Modello Prototipale

A differenza del Modello a Cascata, il Modello Prototipale si focalizza meno sulla redazione della documentazione e più sullo sviluppo del software, questo permette quindi di ottenere la versione finale in meno tempo. L'utilizzo di un prototipo consente inoltre di coinvolgere maggiormente l'utente finale, creando così l'opportunità di avere feedback migliori e più completi e di prevenire le incomprensioni che si possono creare tra il cliente e il team di progetto. Grazie a questo approccio il prodotto finale soddisferà maggiormente le richieste di "*look, feel and performance*" dell'utente.

Tuttavia il Modello Prototipale può presentare anche delle notevoli problematiche. Prima di tutto quando si sviluppa un prototipo si corre il rischio che il cliente non capisca che si tratti di una versione apparentemente funzionante del software, ma che in realtà manca di aspetti fondamentali, della qualità e durevolezza della versione finale. Questo genera nel cliente disorientamento e un attaccamento alla versione mostrata dal prototipo, nel momento in cui deve essere ricostruito il software, che possieda tutte le funzionalità ed un'elevata qualità.

Un altro grosso problema è che spesso lo sviluppatore commette degli errori grossolani pur di mostrare velocemente una versione del prototipo. Per questo vengono scelti dei sistemi operativi o dei linguaggi di programmazioni sbagliati, oppure degli algoritmi insufficienti che compromettono lo sviluppo di tutto il sistema.

Nonostante questi svantaggi, l'utilizzo di prototipi resta un valido aiuto nello sviluppo dei software, la cosa più importante è specificare fin da subito al cliente che esso non è altro che un mezzo per aiutare a recepire nel modo migliore e più completo i requisiti necessari per lo sviluppo di un buon prodotto finale.

2.3 Modello RAD

Al giorno d'oggi pochissime aziende possono permettersi di adottare un modello come quello a cascata che impiega alle volte 6 mesi solo per raccogliere i requisiti. Secondo Burt Rubenstein, vice presidente dei servizi tecnologici al *Cambridge Technology Partners (CTP)*, progetti che durano più di 6/9 mesi rischiano di perdere l'interesse da parte del cliente o di veder modificato lo stesso processo di business. Sempre secondo Rubenstein, nel caso servissero dei miglioramenti o dovessero essere aggiunte delle funzionalità, questo potrebbe sempre essere fatto in una fase successiva.

In quest'ottica si colloca il modello RAD (*rapid application development*), un modello incrementale che si basa su un ciclo di sviluppo molto breve. Lo sviluppo così rapido del software è consentito dall'utilizzo di moduli durante la sua costruzione. Nel caso in cui i requisiti siano stati compresi e sia stato definito lo scopo del progetto, le tempistiche per lo sviluppo del software si riducono nettamente (i.e. da 60 a 90 giorni).

Utilizzando questo modello le varie componenti o funzionalità vengono gestite come dei mini progetti eseguibili in parallelo. Questo permette di svolgere le attività in un tempo limitato e di assemblarle successivamente insieme, in modo da dare al cliente un prototipo funzionante. Sulla base di questo il cliente potrà fornire i propri feedback che serviranno ad apportare i miglioramenti del caso.

In Figura 2.4 viene mostrato un esempio di Modello RAD, nel quale sono definite nel dettaglio le fasi di cui è composto:

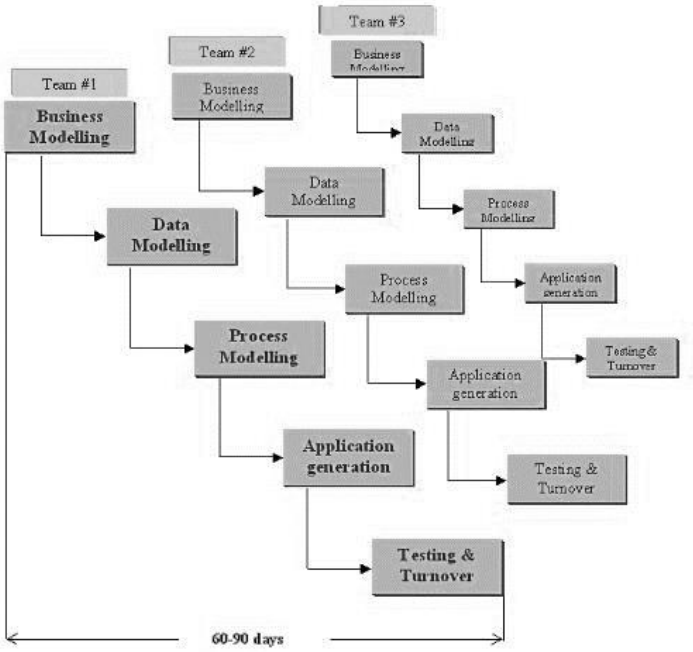

Figura 2.4: *Modello RAD*

Business modeling: Durante questa fase, per prima cosa vengono analizzate tutte le funzionalità del processo di business. Successivamente le informazioni relative alle diverse attività che compongono il processo di business vengono utilizzate per costruire un modello di business.

Data modeling: Le informazioni recepite durante l'analisi del processo di business vengono modellate in più set di dati con caratteristiche ben precise. Sempre in questa fase vengono inoltre definite in maniera precisa le relazioni tra i vari insiemi di dati.

Process modeling: In questa fase vengono utilizzati i set di dati trovati nella fase precedente per costruire il flusso informativo necessario ad ottenere le diverse funzionalità di business. Nel caso debbano essere apportate delle modifiche ai set di dati, esse vengono implementate in questa fase.

Application generation: Durante questa fase viene effettivamente sviluppato il software e conclusa la codifica. Il team di progetto ricorre, quando possibile, a parti di programma già esistenti e qualora queste non siano presenti, ne crea delle nuove

riutilizzabili. In ogni caso gli sviluppatori ricorrono a strumenti automatici per facilitare la creazione del software.

Testing and Turnover: A questo punto vengono svolti tutti i test necessari a verificare lo stato dell'applicativo sviluppato. Tuttavia, dato che molte parti di programma vengono riprese da progetti precedenti, e sono quindi già state testate, questa fase è molto rapida. L'attenzione maggiore deve essere rivolta ai nuovi componenti e alle interfacce presenti.

Questo modello è utilizzabile per quei processi che possono essere scomposti in diversi moduli e per ciascuno dei quali non vengano impiegati più di 3 mesi per il loro sviluppo. Una volta costruiti tutti i moduli necessari dai diversi team, essi vengono uniti insieme per formare il prodotto finale.

La "modularizzazione" del processo non è l'unica condizione per cui è consigliabile l'utilizzo di questo modello. Il RAD infatti è preferibile nei casi in cui si abbiamo risorse altamente qualificate, i rischi tecnici siano bassi e si abbiano tempi di consegna particolarmente stretti.

2.3.1 Pro e Contro del Modello RAD

Per quanto riguarda i vantaggi di questo modello, si è visto che permette uno sviluppo molto rapido e il riutilizzo di componenti già esistenti. Questi motivi, uniti al fatto che si tratta di uno sviluppo incrementale per moduli, consentono maggior facilità di gestione e una riduzione dei costi. Grazie a questo modello inoltre è possibile mitigare i rischi, evidenziandoli e riducendoli fin dalle prime parti del processo. Tutto questo permette di evitare i *failure* catastrofici che invece si possono presentare durante l'utilizzo dei modelli a cascata.

Ovviamente il Modello RAD, oltre a questi aspetti positivi, presenta degli svantaggi:

- Per la riuscita del progetto è necessario un "*commitment*" iniziale, atto a garantirne uno sviluppo rapido e consistente, sia da parte del team di progetto

che da parte dei clienti. Qualora questo *commitment* iniziale dovesse mancare, il progetto sarebbe inesorabilmente destinato a fallire.

- Il modello è applicabile solo a sistemi che possono essere modularizzati. Questo è dovuto al fatto che sarebbe particolarmente difficile, se non impossibile, costruire le varie componenti del software.
- Infine è sconsigliabile l'utilizzo di questo approccio nel caso di rischi elevati. Questi sono dovuti principalmente all'utilizzo consistente di nuove tecnologie nello sviluppo del software e alla necessità di interfacciare il nuovo sistema con numerosi altri programmi.

2.4 Modello a Processo Incrementale

Il modello incrementale non è altro che una combinazione del modello sequenziale con l'approccio incrementale. Ogni volta vengono prodotte delle sequenze lineari che fungono da incremento del software. Ognuna di queste sequenze assomiglia a quelle tipiche del modello a cascata, per questo tale modello a volte è chiamato anche "*modello multi-waterfall*". La prima sequenza non è altro che il "*prodotto core",* il quale soddisfa i requisiti di base, e che deve essere ulteriormente completato da ulteriori funzionalità, alcune delle quali possono essere ancora sconosciute. Questo primo prodotto è mostrato al cliente, il quale sarà così in grado di fornire i feedback che porteranno al raffinamento dei requisiti. Sulla base di questi partirà il piano per uno sviluppo successivo del prodotto, in modo da soddisfare le richieste del cliente e da aggiungere funzionalità e caratteristiche. Questo processo, come è visibile in Figura 2.5, viene ripetuto fino a quando non viene ottenuto il prodotto finale.

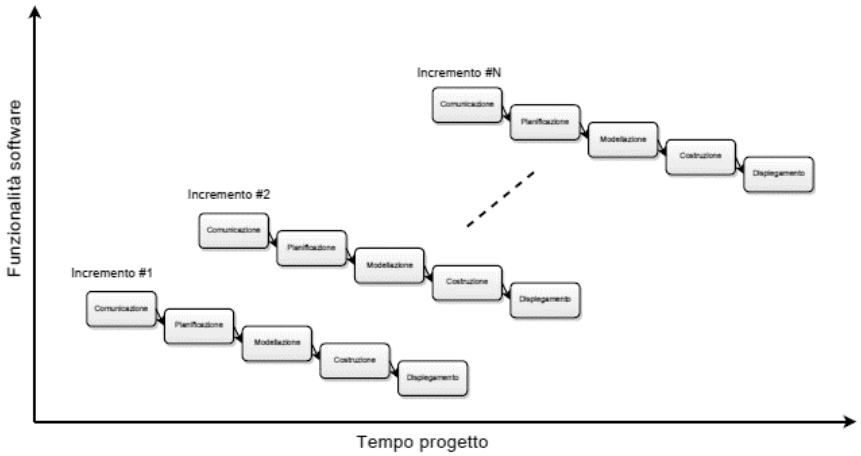

Figura 2.5: *Modello a Processo Incrementale*

Ogni iterazione è costituita dalle seguenti fasi:

- Comunicazione dei requisiti
- Pianificazione
- Modellazione
- Costruzione
- Dispiegamento

Al termine di ogni iterazione viene prodotta una versione del software funzionante, con funzionalità minimali che verranno successivamente ampliate e migliorate.

2.4.1 Pro e Contro del Modello a Processo Incrementale

Il modello incrementale segue un processo iterativo proprio come il modello prototipale. A differenza di questo, però, fornisce ogni volta una versione funzionante, anche se non completa del sistema. Questo, a differenza del prototipo, permette al cliente, non solo di utilizzare il prodotto fin lì ottenuto per valutare la direzione che sta prendendo il progetto e fornire i suoi feedback, evitando le situazioni tipiche dei modelli a cascata, in cui dopo mesi di lavoro il cliente manifesta incertezze sui requisiti precedentemente definiti, ma può anche essere realmente utilizzato.

In questo caso inoltre possono essere utilizzate meno persone nelle fasi iniziali per condurre il progetto, comportando così una diminuzione dei costi. Questo approccio inoltre è utile nel caso in cui non si riesca a portare a termine il progetto entro la scadenza prestabilita in fase di pianificazione. Verranno infatti rilasciate versioni iniziali, ma funzionanti, del software con la possibilità di aggiungere ulteriori funzionalità e caratteristiche nelle fasi successive.

Tuttavia, nonostante questi vantaggi, i metodi incrementali legati al modello a cascata evidenziano alcune problematiche:

- E' sempre necessario avere l'idea generale dei requisiti ai quali l'applicazione dovrà soddisfare, prima di poter dividere il progetto in diverse iterazioni. Di conseguenza, è necessario comunque progettare il software per intero e verificare in una fase successiva quali caratteristiche siano separabili nei diversi incrementi.
- Può accadere che al momento di integrare le varie interazioni si verifichino incompatibilità causate da problematiche che sono sfuggite in fase di pianificazione del progetto, in componenti teoricamente indipendenti tra loro.
- Il cliente potrebbe approfittare della struttura incrementale per avanzare richieste circa l'aggiunta di ulteriori requisiti a cui non aveva pensato inizialmente, andando così a modificare la pianificazione decisa nella fase iniziale del progetto, quando ancora non era avvenuta ancora alcuna divisione del processo. In questa situazione le varie *release* devono essere ripianificate e possono essere necessarie risorse economiche più ingenti di quelle inizialmente stimate.

2.5 Modello a Spirale

La paternità di questo modello si deve al matematico e professore di ingegneria del software Barry W. Boehm, il quale lo definì per la prima volta nel 1986 nell'articolo "A Spiral Model of Software Development and Enhancement." Il modello a spirale è simile al modello incrementale, con un enfasi maggiore sull'analisi del rischio. Esso infatti si basa sulla combinazione del modello a cascata con le sue attività sistematiche, unitamente alla natura iterativa del modello prototipale.

Questo modello iterativo è particolarmente adatto per grossi progetti, con durate dai 6 mesi ai 2 anni, e con un alto livello di complessità. Esso permette uno sviluppo rapido del software, che avviene con rilasci incrementali, i quali aggiungono di volta in volta caratteristiche e funzionalità all'applicativo. Si passa infatti dall'avere un modello di carta o un prototipo, fino alla versione del software completa.

Il modello a spirale è suddiviso in una serie di attività, che vengono ripetute iterativamente come è visibile in Figura 2.6.

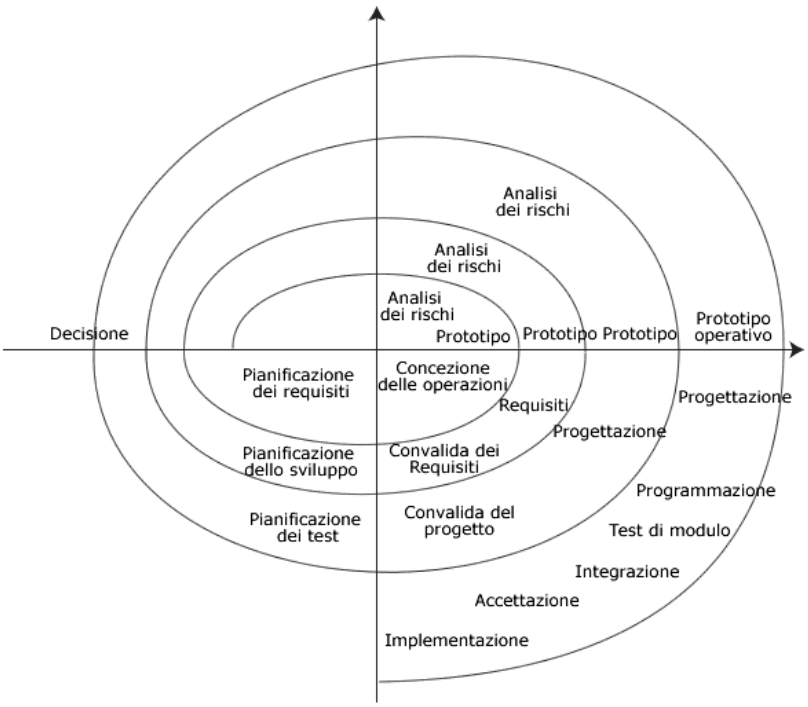

Figura 2.6: *Modello a Spirale*

Le attività vengono fatte partire dal centro della spirale, proseguendo in senso orario e vengono suddivise tipicamente in un numero di regioni che va dalle 3 alle 6. Solitamente si hanno almeno le seguenti fasi: la pianificazione, l'analisi del rischio, la costruzione e test e la valutazione. Ma per un analisi ad un maggiore livello di dettaglio, si riporta una descrizione di tutte e 6 le fasi:

Comunicazione col cliente: Durante questa prima fase vengono svolte tutte le attività che permettono la comunicazione e lo scambio informativo con il cliente.

Pianificazione: In questa fase vengono raccolti i requisiti di sistema e di business del cliente. Vengono inoltre definiti i costi, le risorse, le tempistiche e altre decisioni relative al progetto, come quelle di "*make or buy*" o di riutilizzo di componenti esistenti.

Analisi del rischio: Viene svolto il processo di individuazione e valutazione dei rischi, sia tecnici che di gestione. I rischi possono riguardare la mancanza di esperienza/conoscenza, l'utilizzo di nuove tecnologie, tempistiche troppo stringenti ecc. Si nota come in questo modello il rischio venga tenuto in forte considerazione in quanto questa analisi viene svolta ad ogni iterazione di sviluppo del software. Qualora venissero identificati dei rischi, si dovrebbe procedere all'individuazione delle alternative possibili e alla loro implementazione. Alla fine di questa fase inoltre viene solitamente costruito un prototipo, che verrà preso in input dalle fasi successive.

Ingegnerizzazione: Durante questa fase viene effettivamente sviluppato il software.

Costruzione e test: Una volta sviluppato il software, questo deve essere testato, installato e implementato. Deve inoltre essere fornito al cliente il supporto necessario, sotto forma sia di documentazione che di *training*.

Valutazione: Infine vengono raccolti i *feedback* del cliente relativi allo sviluppo del software e all'installazione vera e propria. Questi *feedback* vengono utilizzati per apportare le modifiche necessarie durante le spirali successive.

All'interno di ogni regione cambia il numero di compiti che vengono svolti. Nei progetti piccoli, nel corso di ogni fase, viene svolto un modesto numero di attività con bassa formalizzazione. Al crescere della grandezza e complessità del progetto anche le attività in ogni regione aumentano in numero e formalizzazione. Tuttavia le così dette "*umbrella activities*", di cui si è parlato all'inizio del capitolo, vengono svolte a prescindere dalle dimensioni del progetto.

Durante le varie iterazioni di questo modello, vengono ripetute le attività viste sopra. Si passa dalla prima iterazione, durante la quale vengono definite le specifiche del prodotto, alle fasi intermedie durante le quali viene sviluppato un prototipo, fino alle fasi finali in cui viene aggiornata di volta in volta la versione del software. Alla fine di ogni iterazione vengono inoltre recepiti i feedback del cliente e il manager del progetto stabilisce il numero di iterazioni che devono essere svolte per completare definitivamente il software.

2.5.1 Pro e Contro del Modello a Spirale

Il modello a spirale ha un approccio realistico allo sviluppo dei software. Esso infatti unisce l'approccio sistematico del modello lineare all'utilizzo di prototipi ad ogni livello del processo iterativo. I prototipi vengono sviluppati fin dalle prime fasi del progetto e fungono da meccanismo per ridurre i rischi.

Questo modello inoltre monitora attentamente i rischi, riproponendone un'analisi ad ogni iterazione. In questo modo i rischi all'interno del progetto vengono tenuti costantemente sotto controllo e ridotti prima che diventino problematici. Nel caso in cui i rischi siano troppo elevati il progetto potrebbe essere dismesso immediatamente, prima di generare inutilmente ulteriori costi.

Tuttavia questo modello, come tutti gli altri, presenta degli svantaggi. *In primis* può essere molto costoso e può essere difficile convincere il cliente della bontà e della controllabilità del processo evolutivo del software. In seconda battuta dipende fortemente dall'analisi e gestione dei rischi. Queste attività richiedono una forte

esperienza e nel caso non venissero svolte correttamente e in tempo utile potrebbero danneggiare in modo compromettente il progetto.

2.5.2 Modello a Spirale WINWIN

Questo modello venne introdotto dallo stesso Barry W. Boehm e si focalizza sulla parte di comunicazione con il cliente che avviene all'inizio del modello a spirale. Durante questa fase, durante la quale il cliente riporta i requisiti che dovrebbe presentare il nuovo software, si instaura solitamente un'attività di negoziazione tra il team di progetto e il cliente per massimizzare le funzionalità e le caratteristiche dell'applicativo rientrando nei costi e nelle tempistiche necessarie.

Boehm ha quindi individuato una serie di attività di negoziazione che devono essere svolte all'inizio di ogni spirale in modo da ottenere una situazione di "vittoria" sia per lo sviluppatore che per il cliente. Da qui prende infatti nome il modello: vittoria (WIN) per il cliente che vede soddisfatte nel miglior modo possibile le specifiche fornite inizialmente e vittoria (WIN) per il team di progetto che può svolgere le attività a costi e in tempi realistici.

Le attività di negoziazione prevedono l'individuazione di tutti gli *stakeholder* e delle loro richieste, l'individuazione dei vincoli e delle condizioni in cui verrebbero soddisfatte solo le richieste di una parte o di nessuna delle due e infine l'attività di negoziazione vera e propria, durante la quale vengono vagiate le alternative possibili, fino al raggiungimento di una vittoria per tutte le parti.

Il completamento di queste attività in modo soddisfacente permette di riuscire a raggiungere la soluzione migliore per tutti gli *stakeholder* e a portare avanti il progetto in maniera ottimale.

2.6 Conclusione

Viene di seguito riportato in Tabella 2.1 un confronto tra i vari modelli visti fin ora, rifacendosi allo studio sintetizzato nell'articolo "*Comparative Study of Prototype Model For Software Engineering With System Development Life Cycle*" del Dr.Rajendra Ganpatrao Sabale:

Model / Features	Waterfall	Prototype	Incremental	Spiral	RAD
Requirement Specifications	Beginning	Frequently changed	Beginning	Beginning	Time-box released
Understanding Requirements	Well Understood	Not well understood	well understood	Well understood	Easily understood
Cost	Low	High	low	expensive	low
Guarantee of Success	Low	Good	High	High	Good
Resource Control	Yes	No	Yes	Yes	Yes
Cost Control	Yes	No	No	Yes	Yes
Simplicity	Simple	simple	Intermediate	Intermediate	very simple
Risk Involvement	High	low	Easily manage	Low	Very low
Expertise Required	High	Medium	High	high	Medium
Changes Incorporated	Difficult	Easy	Easy	Easy	Easy
Risk Analysis	Only at beginning	No risk analysis	No risk analysis	Yes	Low
User Involvement	Only at beginning	High	Intermediate	High	Only at the beginning
Overlapping Phases	No such phase	Yes	No	Yes	No
Flexibility	Rigid	Highly flexible	Less flexible	Flexible	High
Maintenance	Least glamorous	Routine *maintenance*	Promotes maintainability	Typical	Easily maintained
Integrity & Security	vital	Weak	Robust	High	Vital
Reusability	limited	Weak	Yes	Yes	some extent
Interface	minimal	crucial	Crucial	Crucial	minimal
Documentation &Training required	vital	weak	Yes	Yes	limited
Time Frame	Long	Short	Very long	Long	Short

Tabella 2.1

Dopo aver analizzato i modelli tradizionali, averne visto vantaggi e svantaggi, verrà analizzato nel prossimo capitolo il modello Agile, che propone un approccio innovativo alla gestione di progetti di sviluppo del software, in modo da avere un quadro completo per introdurre un caso reale e permettere di trarre le osservazioni del

3 | METODOLOGIE AGILE

Durante gli ultimi decenni il mercato è profondamente cambiato, si è passati da una situazione in cui la domanda superava l'offerta, fino a giungere ad una condizione completamente capovolta. Lungo questo percorso l'attenzione si sposta inevitabilmente sul cliente, il quale diventa l'elemento principale all'interno dell'economia di mercato. Il potere si sposta dalle imprese al consumatore, per il quale non è più sufficiente una sola differenziazione in termini di costi e prodotti, ma diventa necessario un elemento aggiuntivo: la customer experience. Questa si spiega come l'insieme di sentimenti e percezioni, razionali e irrazionali, percepite dal cliente nei momenti di interazione con le aziende ed i loro prodotti.

In quest'ottica e con l'avvento di Internet, che ha contribuito a rendere il mercato ancora più turbolento e soggetto ad una velocità sempre crescente, si è assistito anche nel mondo dello sviluppo dei software, allo spostamento dalla metodologia di tipo waterfall, passando attraverso la metodologia a spirale, fino a giungere allo sviluppo delle metodologie agile. Queste non solo richiedono tempi di rilascio sempre minori, per venire incontro a cambiamenti dei requisiti sempre più frequenti, ma impongono di focalizzarsi sulle volontà del cliente. E' proprio il committente e/o l'utilizzatore del prodotto a guidare il processo di sviluppo, ponendo al centro i suoi requisiti, i quali devono essere soddisfatti nel miglior modo e il più rapidamente possibile. In questo ambito diventano fondamentali le fasi di raccolta delle funzionalità che il nuovo sistema deve avere, le quali si basano sulle percezioni del cliente e sulle conoscenze pregresse dei sistemi fino ad allora in uso. Il successo del progetto infatti non può prescindere dall'impostare fin dalle prime fasi una collaborazione tra team di sviluppo e clienti. Per questo motivo, in questa sezione, oltre ad analizzare le metodologie Agile si vedrà come questi elementi sono presenti e fungono da sostegno all'interno di esse.

3.1 Introduzione all'Agile

Le prime applicazioni di metodologie snelle nei processi di produzione risalgono agli anni 70, quando Taiichi Ohno introdusse per la prima volta in Giappone il Toyota Production System, anche detto Lean Production, il quale si basava principalmente su un sistema produttivo di tipo Pull e la tecnica di produzione Just in Time. Questo sistema produttivo, a differenza dei processi di produzione di massa fino ad allora utilizzati, dava risalto a concetti fondamentali come l'eliminazione degli sprechi e dei sovraccarichi, la semplicità e la flessibilità, il miglioramento continuo e a piccoli passi. Tutto questo doveva avvenire mettendo al centro un valore importantissimo: il rispetto delle persone. Tale sistema infatti, non poteva e non può prescindere dall'avere un personale motivato, autonomo e coinvolto nel miglioramento continuo.

Sulla base di questi principi inizia a svilupparsi la metodologia Agile, la quale si pone in contrapposizione alle metodologie tradizionali, specialmente a quella a Cascata. In un mondo dove è richiesta una velocità sempre crescente nel fare le cose, anche il time to market nello sviluppo dei software si accorcia sensibilmente e aumentano le richieste di modifica dei requisiti anche in fasi avanzate dello sviluppo, i modelli tradizionali iniziano a risultare inadeguati. In questo contesto prende piede il metodo Agile, il cui significato letterale è proprio "caratterizzato da rapidità, leggerezza e facilità di movimento". In questo modo il prodotto finale viene consegnato rapidamente al cliente e il processo di sviluppo avviene in maniera flessibile, fornendo continuamente software funzionante.

All'interno della metodologia Agile si raggruppano iversi metodi innovativi come lo Scrum, l'Extreme Programming, il FDD (Feature Driven Development), il DSDM (Dynamic System Development Method), il Lean Software Development, Crystal Clear e altri ancora. Tutte queste metodologie, indipendentemente dal framework utilizzato, condividono una visione comune e una serie di valori di base. Tutte quante si basano sul concetto di iterazione e danno notevole importanza ai feedback, che servono a raffinare lo sviluppo del software. Esse inoltre utilizzano attività di planning, testing e integrazione continua per migliorare il lavoro di sviluppo. Queste metodologie,

chiamate lightweight, in contrapposizione ai metodi tradizionali, focalizzano particolarmente la loro attenzione sulla collaborazione delle persone, nell'ottica di ottenere un lavoro veloce ed efficace. La filosofia agile ritiene infatti che si debba aver maggior fiducia nelle persone, poiché esse possono aver successo senza un processo di sviluppo formale, cosa non altrettanto vera per il contrario.

In sintesi tutte queste metodologie si basano su di un processo abilitato dalle seguenti proprietà fondamentali:

- **Iterativo e incrementale:** Lo sviluppo del software avviene per iterazioni successive, che si sommano in modo incrementale alle precedenti. Ciascuna iterazione avviene in un lasso temporale ben definito, di breve durata e che deve essere rigorosamente rispettato.
- **Coinvolgimento attivo del cliente:** Durante lo sviluppo si ricorre ad una stretta collaborazione con il cliente, il quale testa ed approva ogni singola iterazione. Questo permette di ridurre i rischi e garantire una soddisfazione maggiore del cliente.
- **Guidato dalle funzionalità:** Viene fatto uno sforzo per fornire le funzionalità richieste, solitamente concentrandosi su quel 20% di funzionalità che verranno utilizzate per l'80% del tempo.
- **Tempo fisso:** Ad ogni iterazione viene fornito un lasso temporale ben definito, durante il quale devono essere svolte tutte le attività.
- **Consegna basata su priorità:** Viene assegnata una priorità alle funzionalità richieste dal cliente durante ciascuna iterazione e vengono consegnate in primis le funzionalità con priorità maggiore.
- **Adattivo:** La metodologia deve essere capace di adattarsi a necessità mutevoli, in modo che l'applicazione possa recepire nuovi requisiti durante lo sviluppo.
- **Responsabilizzazione del team:** I team generalmente sono costituiti da poche persone, in modo da riuscire a comunicare facilmente tra loro. I team stessi

vengono messi in grado di prendere decisioni e gestire autonomamente il progetto.

- **Centrato sulle persone:** Il focus principale è sul lavoro delle persone piuttosto che sui processi. Queste metodologie inoltre danno scarsa rilevanza alla documentazione e ad attività diverse da quelle di sviluppo e testing.
- **Sviluppo rapido:** Lo sviluppo viene svolto velocemente, grazie a moderne tecnologie di sviluppo leggere.
- **Rilasci frequenti:** Al termine di ognuna delle molte iterazioni vengono rilasciate versioni del software funzionanti.
- **Testing:** Viene proposta, quando possibile, una verifica automatica sia del sistema e del suo codice, che dei dati e dei modelli. Se la verifica automatica risulta impossibile, si ricorre ad una verifica manuale.
- **Più disciplina:** Alla base della velocità ci deve essere la disciplina, il gioco di squadra e l'organizzazione del team, le quali consentono di consegnare versioni corrette già la prima volta.
- **Semplicità:** Ogni cosa deve essere mantenuta il più semplice possibile e si deve essere propensi al cambiamento in qualsiasi momento.

Molti dei principi e delle pratiche adottate dalla metodologia agile sono state usate da anni, se non da decenni, in modo individuale nell'ambito dell'Information Technology. La vera innovazione da parte dell'approccio agile è stato il riunirle insieme in un corpus ben strutturato, capace di guidare i team di sviluppo attraverso il processo di pianificazione e rilascio rapido di software.

Nel 2001 un gruppo di 17 progettisti del software e guru dell'informatica si è riunito nella Agile Alliance e si è impegnato al fine di formalizzare questi principi. Il documento finale è stato poi sottoscritto da un gran numero di professionisti, tra i quali erano

presenti anche gli stessi sviluppatori di alcune delle metodologie. Tale documento è conosciuto come l'Agile Manifesto e si basa sui seguenti 4 punti cardine:

1. Gli individui e le interazioni più che i processi e gli strumenti
2. Il software funzionante più che la documentazione esaustiva
3. La collaborazione col cliente più che la negoziazione dei contratti
4. Rispondere al cambiamento più che seguire un piano

All'interno del Manifesto sono inoltre riportati i 12 principi che devono essere seguiti per essere conformi alla metodologia agile:

1) La nostra massima priorità è soddisfare il cliente
   rilasciando software di valore, fin da subito
   e in maniera continua.
2) Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo.
   I processi agili sfruttano il cambiamento
   a favore del vantaggio competitivo del cliente.
3) Consegniamo frequentemente software funzionante,
   con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.
4) Committenti e sviluppatori devono lavorare insieme
   quotidianamente per tutta la durata del progetto.
5) Fondiamo i progetti su individui motivati.
   Diamo loro l'ambiente e il supporto di cui hanno bisogno
   e confidiamo nella loro capacità di portare il lavoro a termine.
6) Una conversazione faccia a faccia
   è il modo più efficiente e più efficace per comunicare
   con il team ed all'interno del team.
7) Il software funzionante è il principale metro di misura di progresso.

8) I processi agili promuovono uno sviluppo sostenibile.
   Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
9) La continua attenzione all'eccellenza tecnica
   e alla buona progettazione esaltano l'agilità.
10) La semplicità - l'arte di massimizzare la quantità
    di lavoro non svolto - è essenziale.
11) Le architetture, i requisiti e la progettazione
    migliori emergono da team che si auto-organizzano.
12) A intervalli regolari il team riflette su come
    diventare più efficace, dopodiché regola e adatta
    il proprio comportamento di conseguenza.

L'obiettivo di tale metodologia è la piena soddisfazione del cliente e non solo l'adempimento di un contratto. Tra gli outcome dati dal suo utilizzo infattisi ha un abbattimento di tempo e costi, a favore di un aumento della qualità.

Durante lo sviluppo di un software si hanno 4 variabili che devono essere gestite: costo, tempo, qualità e funzionalità. Per la prima volta, nel 1994, all'interno della prima edizione del Manuale di DSDM, è stato presentato il modello del triangolo invertito, visibile in Figura 3.1.

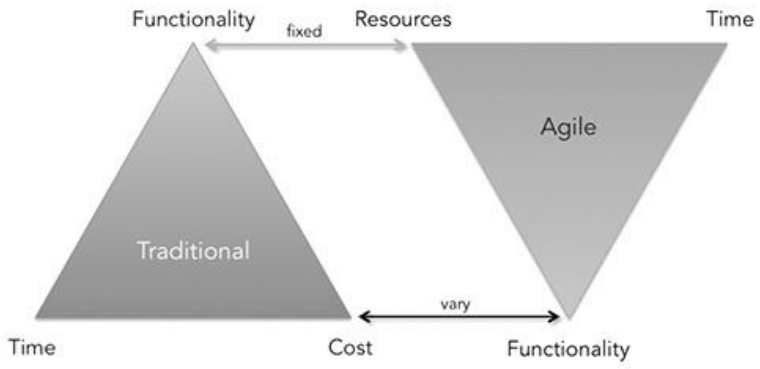

Figura 3.1: *Triangolo di ferro e triangolo invertito*

Questo modello mostra la differenza nel modo in cui vengono legati tempo, costo e funzionalità nelle metodologie tradizionali e in quella agile. Nel caso tradizionale viene fissato l'insieme delle funzionalità e vengono di conseguenza calcolati tempi e costi. Una volta fissate tutte e tre le variabili, in quello che viene definito il triangolo di ferro, è difficile garantire un certo grado di qualità. Spesso infatti si va incontro a un numero maggiore di rischi e ad un conseguente aumento nei fallimenti durante lo sviluppo dei software.

A differenze delle metodologie tradizionali, in quella agile vengono fissati tempi e costi, permettendo all'insieme di funzionalità di variare. Questo non solo consente al team di sviluppo di correre minori rischi, ma permette anche al cliente di modificare i propri requisiti in modo tale da massimizzare l'outcome del processo di sviluppo. Tutto ciò ovviamente permette di aumentare il grado di qualità ottenuto e la soddisfazione di ambo le parti.

A seguito di questa introduzione, dalla quale già si intravedono i vantaggi di questa nuova metodologia, verrà analizzato più nel dettaglio il funzionamento di alcune delle più importanti metodologie agile.

3.2 SCRUM

Tra i frame work dello sviluppo Agile, quello su cui ultimamente c'è una maggiore attenzione è lo Scrum. Esso è nato agli inizi degli anni '90 per gestire prodotti complessi ed è stato presentato per la prima volta nel 1995 da Ken Schwaber e Jeff Sutherland alla conferenza di OOPSALA, durante la quale è stato mostrato ciò che era stato documentato fino a quel momento. Il termine Scrum deriva dal termine inglese che descrive la mischia durante le partite di rugby e proprio da questo prende spunto per descrivere lo sforzo compiuto dallo scrum team di spingere verso la medesima direzione. Scrum non può essere definito come un processo o una tecnica, bensì come

un frame work costituito da ruoli, eventi, artefatti e regole all'interno del quale possono essere utilizzati diversi processi e tecniche. Esso è un frame work light weight, di facile comprensione, ma difficile da adottare. Nato come strumento di gestione di progetti di sviluppo dei software, esso può essere utilizzato in generale come approccio di gestione di qualsiasi progetto. Lo Scrum si basa sulla teoria dell'empirismo, ovvero sul fatto che la conoscenza si basa sull'esperienza e che le decisioni devono essere prese grazie a ciò che si conosce. Per questo, tale frame work utilizza un metodo iterativo e incrementale per gestire i progetti e diminuire i rischi.

I pilastri che supportano tale metodo sono quindi quelli tipici dell'empirismo:

- Trasparenza: Gli aspetti significativi devono essere visibili a tutti e definiti da uno standard comune.
- Ispezione: Devono essere ispezionati frequentemente gli artefatti e l'avanzamento verso gli obiettivi prefissati. Il tutto deve essere fatto senza intralciare il lavoro e da persone competenti.
- Adattamento: Se durante l'ispezione alcuni aspetti non rientrano nei parametri e il prodotto finale deve essere scartato, bisogna fare in modo di modificare il processo e il materiale in tempi rapidi, al fine di garantirne l'accettazione.

3.2.1 Scrum Team

Alla base di questo metodo c'è uno Scrum Team auto-organizzato e cross-funzionale. Questo significa che non solo il team è in grado di autogestirsi, senza essere diretto da nessun elemento esterno, ma che ha al suo interno tutte le competenze necessarie per lavorare al progetto. All'interno dello Scrum Team inoltre si riconoscono ruoli ben precisi, per i quali viene riportata di seguito una breve descrizione:

Product Owner: Si tratta di un'unica persona che dovrebbe possedere autorità, visione del futuro e tempo a disposizione per il progetto. Egli rappresenta il cliente e il business, ed è il responsabile dei requisiti e dell'ordine temporale per la loro realizzazione. Il Product Owner è infatti l'unico responsabile del Product Backlog, una

lista contenente tutti i requisiti che andrebbero sviluppati, e deve assicurarsi che gli elementi al suo interno siano espressi in modo chiaro, siano definiti al giusto livello di dettaglio e siano ordinati nel modo migliore per raggiungere gli obiettivi in modo tale da essere compresi e sviluppati adeguatamente dal Team di Sviluppo. Le decisioni prese dal Product Owner e mostrate nella Product Backlog, devono essere rispettate all'interno dell'organizzazione. Qualsiasi cambiamento nell'ordine delle priorità o di modifica dei requisiti da parte del Team di Sviluppo, deve essere sottoposta all'approvazione del Prodcut Owner, il quale è il responsabile ultimo.

La copertura di questo ruolo può risultare difficile per quanto riguarda il giusto bilanciamento tra il desiderio di micro- gestire un team che dovrebbe essere in grado di autogestirsi e la necessità di mettere a disposizione competenze di aiuto al team richieste dal ruolo.

Scrum Master: Lo Scrum Master è il responsabile del processo, egli si assicura che lo Scrum sia compreso e approvato e che lo Scrum Team aderisca ai valori, alle pratiche e alle regole alla base del frame work. Egli si pone quindi come manager del processo in generale, ma non gestisce e non ricopre un ruolo di autorità nei confronti del team. Egli è un leader al servizio del team e si adopera al fine di rimuovere gli ostacoli che si presentano durante il percorso finalizzato al raggiungimento degli obiettivi. La persona scelta come Scrum Master è sia al servizio del Product Owner, fornendogli consigli, sia al servizio del Team di Sviluppo, agevolandone la creatività e l'empowerment. Inoltre questa figura deve servire da interfaccia tra questi due elementi e porsi da punto di riferimento per gli altri stakeholder per quanto concerne gli aspetti legati allo Scrum.

Team di Sviluppo: Il Team è costituito da professionisti i quali devono produrre un incremento finito alla fine di ogni Sprint. Ciascun Team deve soddisfare le seguenti caratteristiche:

- Auto-organizzato: deve essere in grado di autogestirsi e trasformare nel modo in cui ritiene più appropriato i requisiti del Product Backlog in incrementi funzionanti del prodotto.

- Cross-funzionale: all'interno del team devono essere presenti tutte le competenze necessarie a realizzare un incremento del prodotto.
- L'unico titolo riconosciuto dallo Scrum all'interno del team è quello di sviluppatore
- Non devono esistere sotto-team dedicati a particolari attività come quella di Testing o Business Analysis.
- Ciascun membro ha aree di competenza e focus, ma la responsabilità finale delle decisioni è affidata al team nel suo complesso.

La dimensione del team deve essere compresa tra le 3 e le 9 persone, esclusi Product Owner e Scrum Master. Questo perché team troppo numerosi comporterebbero problematiche dovute al coordinamento, non consentendo al team di rimanere Agile, e team troppo piccoli invece rischierebbero di non avere le skill necessarie a completare il lavoro durante uno Sprint.

3.2.3 Eventi dello Scrum

Lo Scrum è composto da una serie di eventi che permettono di aumentarne la regolarità e sono definiti all'interno di una finestra temporale prefissata. Il cuore di questa metodologia è lo Sprint, di durata predefinita, all'interno del quale si collocano tutti gli altri eventi, che possono essere allungati o accorciati fin quando non viene raggiunto il loro scopo. Il mancato svolgimento di uno di questi elementi all'interno dello Sprint comporta però una diminuzione di trasparenza e il rischio di veder diminuite o addirittura annullate le fasi dedicate all'ispezione e all'adattamento.

In generale, come è visibile in Figura 3.2, si può dire che il metodo si sviluppa partendo dalla creazione di una lista di requisiti (Product Backlog) da parte del Product Owner, che verrà presa come input all'interno di ciascuno Sprint. La lista viene quindi rivista durante la fase di pianificazione (Sprint Planning) da parte del team, al fine di creare un'altra lista composta da un sottoinsieme di requisiti (Sprint Backlog) che verranno

congelati e andranno ad essere implementati all'interno dello Sprint corrente. Durante lo Sprint il team si incontra ogni giorno per verificare il processo (Daily Scrum) e alla fine del periodo prefissato verrà svolta una valutazione retrospettiva (Sprint review e Sprint retrospective). Come output verrà rilasciata una versione funzionante del prodotto, che soddisfa i requisiti scelti in fase di Sprint Planning. Essa verrà mostrata al Cliente ed al Product Owner, per ricevere feedback a riguardo, e in quanto versione funzionante, potrà essere direttamente utilizzata, già al termine di questa fase.

Successivamente verrà ripetuta la fase di Sprint Planning e verranno svolte le attività appena descritte, fin tanto che non saranno stati soddisfatti un numero accettabile di requisiti o non si incorrerà in vincoli di budget o temporali. Qualunque sia il motivo o il momento in cui termina il progetto, il metodo assicura che siano stati sviluppati almeno gli aspetti ritenuti di maggiore importanza.

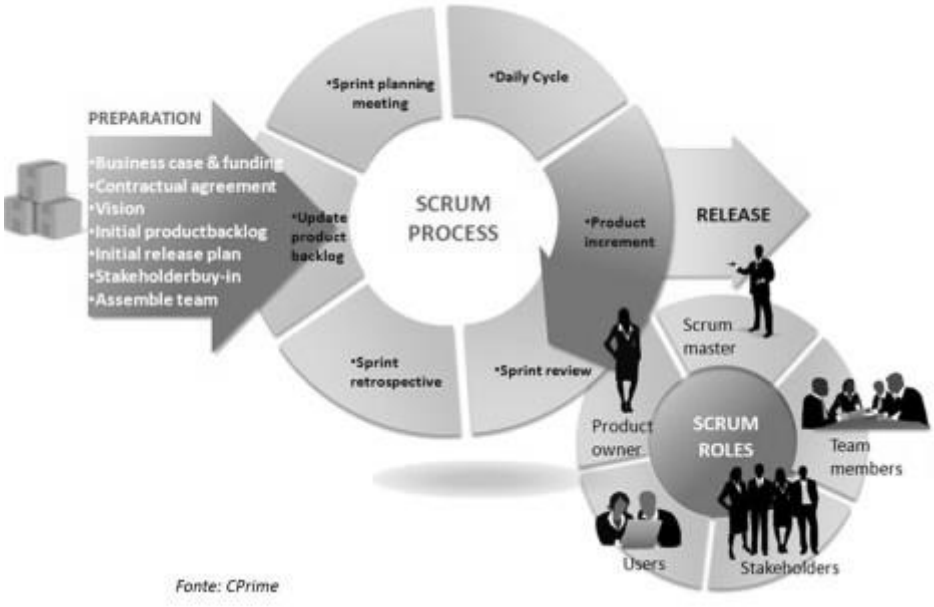

***Figura 3.2**: Pocesso di SCRUM*

Si riporta di seguito una descrizione di dettaglio degli eventi principali dello SCRUM, al fine di consentirne una maggiore comprensione:

Sprint: Lo Sprint è il cuore dello Scrum. Si tratta di un progetto all'interno del progetto e costituisce una guida per realizzare il lavoro. Ha durata massima di un mese e alla fine di ciascuno Sprint si ha il rilascio di una versione di prodotto finita e utilizzabile e si avvia immediatamente lo Sprint successivo. Come visto sopra, ciascuno Sprint è composto da: Sprint Planning, Daily Scrum, lavoro di sviluppo, Sprint Review e Sprint Retrospective. Durante questo evento non possono essere fatte modifiche che comportano rischi per lo Sprint Goal e qualsiasi variazione di Scope deve essere discussa dal Product Owner e Team di Sviluppo.

Sprint Planning: Si tratta di un incontro di una durata massima di 8 ore per Sprint di un mese e inferiore per Sprint più brevi. Lo Scrum Master si pone come guida di questa attività e si assicura che tutti i partecipanti, Team di Sviluppo, Product Owner e rappresentanti della clientela, ne comprendano le finalità. Durante lo Sprint Planning devono essere definiti lo Sprint Goal, cosa può essere consegnato nell'Incremento sulla base del Product Backlog, degli Incrementi precedenti e delle capacità del Team e come fare per svilupparlo.

Daily Scrum: E' un evento della durata massima di 15 minuti che serve a definire le attività da svolgere per le seguenti 24h. Durante ciascun incontro ogni elemento del Team di Sviluppo descrive le attività che ha fatto e che farà per raggiungere lo Sprint Goal e la presenza di ostacoli durante questo percorso. In questo caso il Team di Sviluppo è responsabile dell'attività e solo i suoi membri vi possono partecipare.

Sprint Review: Questo incontro viene tenuto alla fine di ciascuno Sprint per ispezionare l'Incremento e modificare, se necessario, il Product Backlog. A questa riunione partecipano sia lo Scrum Team che tutti gli altri stakeholder ed è finalizzata principalmente a recepire feedback e commenti sull'incremento e a sviluppare la collaborazione. L'incontro ha una durata di 4 ore per Sprint di un mese ed inferiore per Sprint più brevi e anche in questo caso lo Scrum Master si pone come responsabile dell'attività.

Sprint Retrospective: Si tratta di un'occasione per lo Scrum Team per ripercorrere il lavoro svolto, al fine di individuare possibilità di miglioramento, al termine della Sprint Review e prima del successivo Sprint Planning. Questo incontro ha una durata massima di 3 ore per Sprint di un mese e anche in questo caso la responsabilità è affidata allo Scrum Master.

3.3 Extreme Programming

Tra le più note e controverse metodologie Agile di sviluppo del software si ha l'Extreme Programming, spesso abbreviato come XP, che venne formulato per la prima volta nel 2000 da Kent Beck. Questa metodologia deve il suo successo al fatto di conferire una forte centralità alla soddisfazione del cliente, fornendogli il software desiderato al momento desiderato. Questa metodologia permette infatti agli sviluppatori di rispondere in modo appropriato a qualsiasi cambiamento nelle richieste del cliente, in qualsiasi fase di sviluppo del prodotto. Si viene a creare un forte lavoro di squadra tra team, manager e clienti, che permettere di svolgere il lavoro nel modo più produttivo possibile. L'Extreme Programming permette di completare i progetti tra i 6 e i 15 mesi e utilizza piccoli team di lavoro, composti da un numero di persone compre tra 2 e 12.

L'XP si basa su 4 valori fondamentali, che si pongono come guida durante tutto il processo di sviluppo:

- *Comunicazione:* Si tratta di un elemento fondamentale in quanto i clienti devono comunicare i requisiti agli sviluppatori, i quali a loro volta forniscono idee e soluzioni strutturali. Questo consente di favorire lo sviluppo di un prodotto il più coerente possibile con quanto desiderato e ridurre i problemi relativi alla mancanza di scambio informativo.
- *Semplicità:* Questa caratteristica è valida sia per quanto riguarda i sistemi, che devono essere semplici ma funzionanti, sia per quanto riguarda la metodologia stessa che deve essere facilmente utilizzabile e produrre il minimo numero di

artefatti (lista dei requisiti, documento di pianificazione e codice). Semplificare le cose permette di rispondere in modo migliore ai cambiamenti e al rischio, senza andare a vincolarsi in soluzioni complicate.

- *Feed-back:* Questi possono provenire sia dal collaudo delle unità, fatto durante la codifica dagli sviluppatori, che dai test funzionali, fatti dal cliente sui rilasci frequenti del team. Questi permettono un duplice vantaggio: sia a livello qualitativo, che a livello temporale. Nel primo caso è importante per capire cosa funziona e cosa no, nel secondo invece sia un miglioramento dovuto al fatto di collezionare feedback il più presto e spesso possibile. In questo modo i problemi risultano essere minori, così come i costi per risolverli.
- *Coraggio:* Tale qualità è richiesta sia agli sviluppatori che ai clienti. Nel primo caso perché gli sviluppatori, utilizzando l'XP, vanno contro le metodologie tradizionali comunemente utilizzate nello sviluppo del software. Nel secondo caso invece è dovuto al fatto di richiedere al cliente una collaborazione e un coinvolgimento crescente rispetto al passato.

Nello sviluppo dei software esistono vari tipi di rischio. Tra questi si hanno il cambiamento dei requisiti, cambiamenti nel problema, cambiamenti nel mercato o anche solo una comprensione parziale o una scarsa definizione dei requisiti. L'XP, a differenza di altri metodi, favorisce il cambiamento, in quanto il cliente ha sempre il diritto di cambiare i requisiti. Il cliente infatti prende attivamente parte al progetto definendo i requisiti, collaborando col team di sviluppo e fornendo feedback continui sulle versioni rilasciate. Il processo che descrive l'Extreme Programming è infatti pensato in modo tale da consentire cambi di rotta in qualsiasi momento.

3.3.1 Processo XP

Il metodo XP prevede un processo incrementale e iterativo. Il progetto viene diviso in sotto-progetti in modo da avere rilasci frequenti, così da poter ottenere un numero maggiore di feedback. Come è visibile in Figura 3.3 i rilasci sono pianificati durante il

Planning Game. In questo caso o il cliente definisce cosa deve essere rilasciato e spetta agli sviluppatori indicare i tempi di lavoro, oppure il cliente fornisce un orizzonte temporale delimitato e saranno gli sviluppatori a decidere la quantità di lavoro

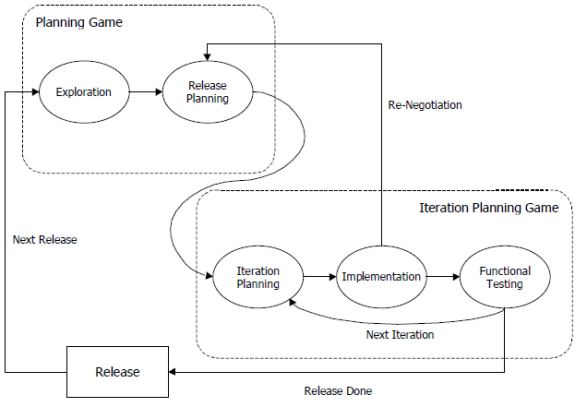

Figura 3.3: *Processo XP*

effettivamente realizzabile. Ciascun ciclo di rilascio sarà poi costituito da un certo numero di iterazioni, come si può vedere nell'Iteration Planning Game.

Il Planning Game è composto dalle seguenti tre attività:

- *Esplorazione:* Il cliente scrive la User Story su degli appositi cartoncini, in modo da indicare i requisiti che il sistema dovrebbe avere. Spetta poi agli sviluppatori indicare le tempistiche di sviluppo di quanto desiderato dal cliente.
- *Pianificazione:* Si tratta della fase di negoziazione tra cliente e sviluppatori, per decidere il numero di funzionalità che possono essere sviluppate dati tempi e risorse. Le storie vengono ordinate secondo priorità e si va a creare il Commitment Schedule con le funzionalità da implementare durante un dato rilascio e le relative tempistiche.
- *Sterzata*: Durante questa fase vengono introdotte delle modifiche a quanto negoziato, a seguito di quanto sta effettivamente emergendo.

Dopo la fase di pianificazione inizia la vera e propria fase di sviluppo, in cui il Commitment Schedule viene diviso in più iterazioni. Durante ciascuna iterazione ha luogo un Iteration Planning Game, composto dalle stesse fasi del precedente. In questo caso però durante la fase di *Esplorazione* vengono decise le funzionalità da realizzare, scegliendole dallo Schedule. Sempre in questa fase, ciascuna storia scelta viene a sua volta suddivisa in task ingegneristici. Ciascun task viene poi scelto volontariamente nella fase di *Pianificazione* da uno sviluppatore che si preoccupa anche di fornire una stima dei tempi di lavoro. Tutti questi dati, nome dello sviluppatore, tempistiche e task, concorrono a formare l'Itineration Schedule. In fine nella fase di *Sterzata* si passa alla codifica vera e propria da parte degli sviluppatori.

3.3.3 Pratiche e regole dell'XP

L'XP si basa sua una serie di pratiche e regole che lo differenziano in maniera sostanziale dai metodi tradizionali, la cui forza non sta tanto nei singoli precetti, quanto nel loro utilizzo congiunto, che porta a bilanciare debolezze di ciascuno, con i punti di forza degli altri.

Le pratiche e le regole dell'XP, che verranno riportate in seguito possono essere suddivise in gruppi:

- **Management**
  - *Metriche:* L'XP si basa su due metriche principali: il Load Factor e lo score del Functional Test suite. Il primo è il rapporto tra i giorni ingegneristicamente ideali (giorni in cui uno sviluppatore riesce a lavorare completamente in disturbato) e gli effettivi giorni del calendario. Esso descrive la velocità di lavoro da parte del team. L'altro indicatore descrive qual è il progresso del progetto. Una suite non è altro che l'insieme dei test, concordati tra clienti e sviluppatori, che devono essere svolti per le varie funzionalità presentate attraverso le

storie raccolte dai clienti. L'incremento del numero di suite e il miglioramento del relativo score descrive la velocità del progetto e fornisce un indicatore al cliente sull'adempimento di ciò che ci si aspetta dal sistema.

- *Ruoli:* Anche nel XP come nello SCRUM vengono definiti una serie di ruoli. Tra questi si ha il cliente, che definisce, attraverso le storie, i requisiti e i test del sistema e il programmatore, che non utilizza specialisti come analisti e architetti del software, ma si occupa di tutti gli aspetti. Il ruolo di management viene diviso in due distinte figure: il coach che si occupa dell'avanzamento tecnico e della sua esecuzione e il tracker che si occupa di raccogliere gli indicatori e verificare la corrispondenza con quanto schedulato. Infine si hanno il tester, che aiuta i clienti a scrivere i test per le funzionalità descritte e viene solitamente ricoperto da uno sviluppatore o dal tracker, e il consulente il quale serve solamente ad infondere conoscenze e unicamente nei casi in cui servano maggiore competenze tecniche rispetto quelle già possedute dagli sviluppatori.

- *Luogo di lavoro e strumenti:* Nell'XP i team sono solitamente composti da 2 a 12 persone e lavorano tutti insieme in un'unica grande stanza. Gli strumenti utilizzati servono principalmente ad agevolare il refactoring, di cui tratteremo in seguito, e facilitare i test. Per questo vengono usati tool come il Refactoring browser e il Testing Framework.

- *Standup meeting:* Si tratta di una riunione tenuta ogni giorno per un paio di minuti, in cui ciascun membro del team presenta il suo lavoro e i problemi incontrati.

- *40 ore la settimana:* I progetti di XP hanno la prerogativa di impegnare il personale del team per sole 40 ore la settimana. Questo significa che lavorare per un periodo più lungo porterebbe inefficienze per il team e il suo lavoro, invece che apportare vantaggi.

- **Sviluppo**

  - *Ciclo di sviluppo:* L'XP prevede che per la fase di implementazione avvenga a piccoli passi e seguendo una procedura ben precisa, che deve impiegare qualche ora o al massimo una giornata. Durante questa procedura ciascuno sviluppatore deve analizzare cosa deve essere fatto, individuare i test da fare, implementare, testare ed apportare eventuali semplificazioni di codice e modifiche.

  - *Integrazione continua:* Durante questa metodologia le modifiche vengono integrate frequentemente, in modo da evitare errori maggiori in fasi successive. Il tutto avviene in modo sequenziale da parte di un unico sviluppatore che apporta le sue modifiche, testa e rilascia il nuovo sistema.

  - *Collective code ownership:* Questa pratica implica che il codice sia responsabilità di tutti i membri del team e non di un unico individuo. Ciò viene fatto in modo tale che ciascun membro del team sia in grado di apportare modifiche, qual ora sia necessario.

  - *Programming in pairs:* Durante la produzione di codice gli sviluppatori devono lavorare in coppie. Questo comporta che uno dei due sviluppi effettivamente il codice e l'altro funga da revisore, in modo da diminuire il numero di errori e mantenere attiva l'attenzione dell'altro. Il revisore svolge anche la funzione di pensare anticipatamente a soluzioni di problemi, ai test che possono essere fatti ecc. Le coppie non vengono fissate inizialmente e anche al loro interno spesso vengono scambiati i ruoli.

  - *Coding standars:* Il team stesso o un qualche elemento esterno definisce le regole da utilizzare durante la codifica e che devono essere rispettate dal team durante tutto il progetto.

- *Cliente on*-site: Durante quasi tutte le fasi del progetto dovrebbe essere presente un cliente. Questi dovrebbe essere possibilmente un esperto e un potenziale utilizzatore del sistema. La presenza del cliente dovrebbe creare valore e diminuire i rischi grazie alla presenza di feedback continui.

- *Testing inarrestabile:* I test sono una parte indispensabile nella metodologia XP. Essi servono a verificare l'integrità del sistema in seguito ad una modifica e ad assicurarsi che il progetto stia seguendo la strada tracciata dal cliente. I test, sia che siano di unità che di funzionalità, devono essere il più possibili automatizzati, in modo da diminuire il lavoro umano.

- **Design**

A differenza degli altri modelli l'XP non utilizza il design dell'up-front, ovvero prima dell'implementazione non viene definito il sistema in termini di architettura e design. Vengono invece utilizzate delle pratiche come il Refactoring e il System Metaphor che verranno descritte in seguito.

- *Do the simplest thing that could possibly work:* Gli sviluppatori nella scelta di cosa implementare dovrebbero ricadere sull'alternativa più semplice possibile. Questa deve essere la più facile da inserire nel sistema esistente, così come la più facile da usare ecc. L'alternativa scelta inoltre deve essere funzionante. Benché questo non possa essere effettivamente stabilito prima dei test, lo sviluppatore deve esserne abbastanza certo.

- *You are not gonna need it:* Questa pratica definisce il fatto che lo sviluppatore debba implementare solo ciò che è necessario in quel momento, resistendo alla tentazione di implementare anche cose che potrebbero servire in futuro.

- *Refactor Mercilessly:* Il refactoring è una tecnica usata per cambiare la struttura interna, come i nomi, l'ordine di unità di codice e le loro dipendenze, senza però modificarne il comportamento esterno. Questo viene utilizzato dall'XP solo nel caso in cui venga migliorato il sistema, in modo tale che sia facile modificarlo, senza andare però ad impattare sul suo comportamento. Il Refactoring inoltre deve essere utilizzato "mercilessly", ovvero ogniqualvolta vada a semplificare in qualche modo la struttura. Esso infine apporta veramente vantaggi solo se utilizzato con altre pratiche dell'XP come il Relentless Testing e il Collective Code Ownership.

- *CRC (Class Responsibility Collaborator) Session:* Si tratta di un metodo che viene utilizzato per visualizzare la struttura del Sistema e serve come incentivo alla discussione al team di sviluppo. Esso si serve di cartellini sui quali vengono riportati in alto il nome della classe, nella colonna di sinistra le responsabilità e in quella di destra il nome delle altre classi con cui deve collaborare per assolvere alla responsabilità descritta a sinistra. I cartellini vengono quindi disposti in modo tale che quelli più vicini tra loro abbiano un legame maggiore e vengono poi spostati per visualizzare diverse architetture di sistema.

- *System Metaphor:* Il sistema si basa su una serie di metafore che fungono sia da fondamenta per il design del sistema che da linee guida per nominare le varie classi, metodi ecc. Queste metafore servono inoltre a facilitare la comunicazione tra cliente e sviluppatori e nel caso non risultino efficaci devono essere migliorate o cambiate.

- *Lazy Optimization:* Questo semplice principio prevede che l'ottimizzazione debba essere tenuta come ultima spiaggia. Ciò è coerente con il detto, tipico delle metodologie Agile, "Make it work.Make it right. Make it fast".

3.4 Feature Driven Development

Il metodo FDD fu introdotto per la prima volta da Jeff De Luca e Peter Coad nel 1997 e fù successivamente formalizzato nel 1999 nel libro *Java modelling in color with UML.* Anche in questo caso l'obiettivo della metodologia è quella di fornire un prodotto funzionante in fasi ripetute il più frequentemente possibile. L'FDD si basa sull'utilizzo di best practice guidate dalle funzionalità richieste dal committente. Questo processo, altamente adattivo, prevede:

- Un gran numero di iterazioni molto veloci
- L'enfasi sulla qualità
- Delivery funzionanti ad ogni iterazione
- La soddisfazione di clienti, manager e sviluppatori.

Il metodo FDD è costituito da 2 fasi principali: la prima riguarda la raccolta di tutte le feature da sviluppare, la seconda invece riguarda la loro implementazione, una di seguito all'altra, nell'arco temporale di 2 settimana circa. Questo metodo infatti scompone il problema generale in sotto-problemi indipendenti tra loro, i quali permettono di ridurre la necessità di comunicazione. Questa, benché indispensabile all'interno di un progetto di sviluppo di software, porta ad aumentare la complessità e quindi la possibilità di avere problemi. La divisione in piccoli problemi permette inoltre di scoprire in anticipo eventuali errori e quindi ridurre i costi relativi alla loro risoluzione. Questo beneficio è dato dal fatto di ravvicinare temporalmente la fase di analisi a quella di testing in ciascun ciclo iterativo.

3.4.1 Ruoli nell'FDD

Nelle metodologie Agile, l'aspetto più importante sono le persone. In questo caso vengono definiti 6 ruoli principali:

- *Chief Architecture:* Questa figura è responsabile sia per il design del sistema che per il team nella fase di elaborazione dell'architettura del sistema. Per questo motivo egli non deve solo avere skill di tipo tecnico, ma deve essere anche capace di porsi come guida.

- *Developmente Manager:* Egli è il responsabile delle attività quotidiane ed ha l'autorità di risolvere eventuali conflitti nell'assegnazione di risorse.

- *Chief Programmer:* Si tratta di programmatori dotati di una certa esperienza nel campo di sviluppo dei software che lavorano nell'analisi dei requisiti e nel design ad alto livello. Essi sono inoltre responsabili di piccoli team che si occupano di analisi e design ad un livello più basso. L'aumento del numero di Chief Programmers aumenta la velocità del progetto, grazie alle loro skill, esperienze e qualità personali che li portano ad essere più produttivi degli altri.

- *Class Owner:* Sono sviluppatori sotto la responsabilità dei Chief Programmers all'interno di piccoli team e che si occupano di tutte le fase tipiche dello sviluppo di feature del nuovo sistema. Benché l'FDD organizzi il lavoro in feature, ognuno di loro è responsabile dello sviluppo e implementazione di una classe. Questo permette non solo di accrescere il senso di responsabilità personale, ma permette anche maggiore coerenza per ciascuna classe.

- *Domani expert:* Queste figure vengono scelte tra gli utenti, gli sponsor e gli analisti funzionali e fungono da base di conoscenze su cui si basa il progetto. La loro partecipazione, dai requisiti ai feedback, è fondamentale per il corretto sviluppo del sistema.

Dato che l'FDD organizza le attività progettuali per feature, è importante descrivere come funziona il *Feature Team.* Viene inizialmente affidata una funzionalità a ciascun

Chief Programmer, il quale a sua volta individua un Class Owner coinvolto nella consegna di quella feature. Ciascun Class Owner però può lavorare con più team di sviluppo. Per ciascuna feature viene infatti creato un gruppo ad-hoc, per la durata di non più di due settimane, che si occuperà di svolgere il lavoro. Tale gruppo prende appunto il nome di Feature Team e i membri all'interno di ciascun gruppo possono variare in seguito al design o alla costruzione della specifica funzionalità.

3.4.2 Processo FDD

Come visto l'FDD è una metodologia iterativa e incrementale che viene divisa in 5 fasi principali, definite *processi,* in cui le prime tre fasi vengono svolte in modo sequenziale e hanno validità per tutto il modello, mentre le ultime due vengono iterate ripetutamente per ciascuna feature, come è visibile in Figura 3.4.

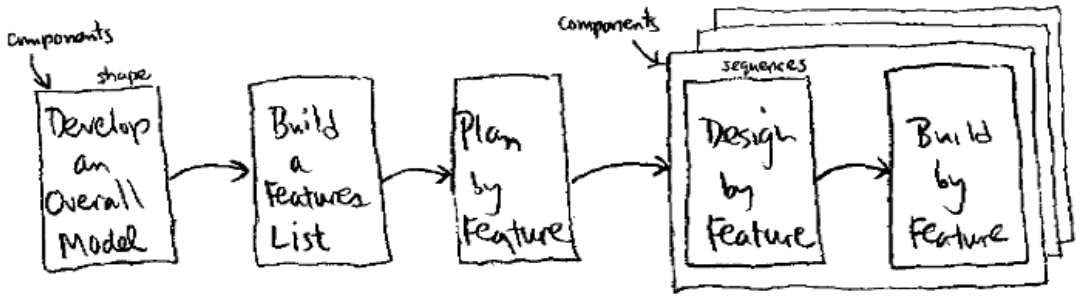

***Figura 3.4**: I 5 processi dell'FDD*

Di seguito si riporta una descrizione di ciascuna di queste 5 fasi:

1) **Sviluppo di un modello generale:** Durante questo processo i domain expert collaborano con gli sviluppatori per individuare lo scope e il contesto del sistema ad alto livello, per poi andare a definire lo scheletro del modello da seguire. Successivamente i domain expert dettagliano il problema in aree circoscritte, in modo tale che il lavoro venga diviso tra sotto-team che si occupano di elaborare un modello per ciascuna area e presentarlo ai colleghi. Alla fine di questa fase viene eseguito un merge dei vari modelli, riportandosi

ad un livello superiore, e preoccupandosi di aggiornarlo lungo l'intero itinerario.

2) **Definizione di una lista di priorità:** A questo punto un team composto solitamente dai soli Chief Programmer, si occupa di suddividere funzionalmente il dominio sopra individuato in diverse aree, seguendo la suddivisione data dai domani expert, e di esploderlo ulteriormente in un insieme di feature set. A questo punto l'insieme di feature che compongono ciascun set vengono prioritizzate, pesate e ordinate gerarchicamente.
3) **Pianificazione per funzionalità:** In questa fase si passa a redigere il piano di sviluppo. In base alla lista redatta al punto precedente, al carico di lavoro all'interno del team di sviluppo e la complessità delle feature, viene dato un ordine di implementazione alle varie funzionalità e vengono individuate delle milestone per le iterazioni delle due fasi successive. All'interno di questo processo devono essere inoltre definiti: per ciascuna feature, un chief programmer e una data di completamento e per ciascuna classe, un class owner.
4) **Progettazione per funzionalità (Design by Feature, DBF):** A questo punto il chief programmer seleziona all'interno del pacchetto di feature che gli sono state assegnate, quelle che devono essere sviluppate in un paio di settimane. Dopodiché egli individua le classi che dovrebbero essere coinvolte nella scelta delle funzionalità e si occupa di contattare i relativi class owner e formare i team di sviluppo. Il team elabora i diagrammi di sequenza, dettagliati per ciascun elemento, che vengono usati dal chief programmer per raffinare il modello generale, mentre il class owner si occupa di scrivere le classi e gli header.
5) **Sviluppo per funzionalità (Build by Feature, BBF):** Basandosi su quanto uscito dalla progettazione per funzionalità, viene iniziato lo sviluppo di una feature utile al cliente. Ciascun class owner iniza la produzione di codice per le classe di competenza, il quale viene poi successivamente sottoposto ad uno unit test e all'ispezione di codice. Se l'esito di queste verifiche, effettuate dal chief

programmer e dal team funzionale, risulta positivo, il codice viene considerato eligible per la costruzione.

L'FDD inoltre fornisce al team il tracciamento del progresso attraverso due punti chiave:

- Conoscenza del numero totale di feature alla fine della prima fase
- Definizione, a grandi linee, delle percentuali che ciascuna fase impiega:

Sviluppo di un modello generale	15%
Definizione di una lista di priorità	5%
Pianificazione per funzionalità	8%
DBF e BBF	77%

Tabella 3.1: Percentuale processi FDD

Come si nota in Tabella 3.1 la parte più rilevante è costituita dal DBF e dal BBF, che vengono infatti suddivisi a loro volta in sei milestone, di cui vengono a loro volta definite le percentuali. Ovviamente queste varieranno, entro certi limiti, in base alla situazione e agli sforzi sostenuti durante il progetto, ma a grandi linee saranno quelle riportate in Tabella 3.2, ed essendo che il DBF e BBF avvengono nell'arco di due settimane, ciascuna fase di cui sono composte avverrà in tempi brevissimi.

DBF			**BBF**		
Walkthrough the domain	Design	Inspect the design	Code/Test	Inspect the code	Promote to build
1%	40%	3%	45%	10%	1%

Tabella 3.2: Milestone di DBF e BBF

Per ogni feature appartenente ad un feature set può essere individuata la percentuale di completamento. Si può quindi dire ad esempio, che una funzionalità che arriva nella fase di coding è completa al 44%. A sua volta può essere rilevato lo stato di completamento per ciascuna area e per il progetto intero. In questo modo possono essere riportate il numero di feature non ancora iniziate, quelle in progress e quelle già completate.

3.5 Dynamic System Development Method

Agli inizi degli anni ’90 iniziò a prendere piede il termine RAD, Rapid Application Development, che mostrava il malcontento degli utenti e del mondo IT verso i metodi tradizionali, che si approcciavano in modo inefficace ad un ambiente in movimento e alla necessità di dar sempre maggiore spazio alla voce dei clienti. In questo contesto nacque nel 1994 il Consorzio DSDM, i cui 16 fondatori avevano in mente l’obiettivo di sviluppare e promuovere un framework RAD. Questo framework venne successivamente sviluppato negli anni, basandosi però sempre sui principi di base e imponendosi come soluzione valida nello sviluppo veloce di sistemi, sia in ambito tecnologico che di business. Nel 2001 uno dei fondatori del Consorzio DSMD partecipò alla redazione e sottoscrizione del Manifesto Agile ed è proprio nell’ottica Agile che si pone il metodo DSDM nello sviluppo di software. La stessa Mission Statement del Consorzio DSDM recita come segue: “*The mission of the DSDM Consortium is to develop and promote best practice and learning for successful Agile projects”.* Di seguito si riporta come gli stessi 9 principi, alla base del DSDM, si ricolleghino ai 12, precedentemente citati, del Manifesto. La violazione di uno solo di questi principi porta a contraddire la filosofia del DSDM, incrementando notevolmente il rischio del progetto, contrapponendosi invece ai passi della struttura DSDM del progetto, che possono essere modificati o saltati all’occorrenza.

3.5.1 I 9 Principi

1) **Partecipazione attiva dell'utente**: Questo è il principio più importante di tutto il metodo, in quanto permette di diminuire errori nella raccolta dei requisiti e di conseguenza di ridurre i costi. Durante tutto il processo di sviluppo devono essere scelti continuamente piccoli gruppi di utenti, che dovranno collaborare con il team, per la riuscita del progetto.

2) **Autonomia decisionale del team**: Al fine di diminuire errori di comunicazione, rallentamenti nei tempi e incrementi nei costi, devono essere diminuite strutture verticali, a vantaggio dell'empowerment nei confronti dei membri del team.

3) **Rilasci frequenti del prodotto**: Il fatto di avere delivery frequenti permette di percepire gli errori più velocemente e di risolverli con maggior facilità, essendo vicini alla sorgente d'errore.

4) **Rilasciare le versioni dando priorità allo sviluppo delle funzionalità finalizzate al business**: In primo luogo devono essere soddisfatti i bisogni del business e solo in un secondo momento bisogna andare ad occuparsi delle altre funzionalità. Inoltre il metodo parte dall'idea che è convenite rilasciare le versioni il prima possibile, in quanto l'80% del prodotto può essere sviluppato nel 20% del tempo di sviluppo del prodotto intero. Nel tempo rimanente potranno poi essere applicati i miglioramenti del caso e corretti gli errori segnalati dagli utenti.

5) **Sviluppo iterativo e incrementale:** Al fine di diminuire la complessità del progetto, questo deve essere suddiviso in vari incrementi che devono essere iterati finché le feature sviluppate non riescono a ricoprire tutti i requisiti sottoposti dal committente. Gli incrementi devono essere i più piccoli possibili, in modo tale che sia più facile rispondere a cambiamenti nei requisiti, senza andare ad impattare sulla bontà del progetto.

6) **Tutti i cambiamenti effettuati durante lo sviluppo devono essere reversibili:** A causa dei cambiamenti nelle priorità dei requisiti il sistema deve essere flessibile e dinamico. Per farlo deve essere supportato da moderni software tool. Il fatto di suddividere lo sviluppo in piccoli incrementi permette inoltre di avere piccole perdite del lavoro precedentemente svolto qual ora si verifichino cambiamenti.

7) **Le specifiche e i requisiti sono definiti ad alto livello:** Durante la fase di ricerche lato business devono essere definiti dei requisiti di alto livello, che verranno congelati, in modo da limitare il numero di richieste di cambiamenti durante il processo.

8) **La fase di test è integrata nel ciclo di vita del prodotto:** A differenza di molte metodologie che prevedono la fase di test successivamente a quella di progettazione e implementazione, questo metodo prevede la fase di test in momenti precedenti nel processo di sviluppo.

9) **Collaborazione e cooperazione tra gli attori coinvolti:** Questa deve essere obbligatoria tra i membri del team tecnico e di quelli lato business, in modo da ottenere un ambiente in cui sia possibile raccogliere facilmente i requisiti in primis e i feedback a prodotto completato.

Questi 9 principi, non solo sono riconducibili ai 12 del Manifesto Agile, ma si ricollocano perfettamente anche nei 4 valori chiavi presentati dal Manifesto stesso. Essi inoltre devono essere pedissequamente rispettati nell'applicazione del metodo DSDM, che risulterà più o meno agile a seconda della situazione e dei vincoli che si presenteranno di progetto in progetto.

3.5.2 La struttura del progetto e le tecniche chiave utilizzate

Il progetto DSDM segue 7 passi che si appoggiano su una serie di ruoli e figure. All'interno del metodo infatti viene riconosciuto un team IT, all'interno del quale non vengono fatte distinzioni tra sviluppatori, progettisti e analisti, con la sola eccezione dei tester e in cui deve essere presente un team leader. Oltre al team IT vengono individuati il manager del progetto, che si pone come interfaccia tra team tecnico e utenti, e il coordinatore tecnico, che come suggerisce il nome, serve a coordinare aspetti tecnici come quelli relativi all'architettura e alla qualità. Infine sono presenti gli utenti che collaborano al progetto, che vengono divisi in ambasciatori e consiglieri a seconda che partecipino rispettivamente a tempo pieno o part-time. Oltre a queste figure, nell'adottare il metodo DSDM, devono essere presenti due ulteriori figure part-time: il "*visionary user*" che si occupa di motivare il team e di assicurare la coerenza tra gli obiettivi del progetti e quelli di business e "*l'executive sponsor*" che si occupa delle risorse da impegnare nel progetto.

Grazie al lavoro di queste figure possono essere seguiti i 7 passi di cui si compone il metodo, i quali non devono essere necessariamente svolti al fine di seguire il DSDM. Ciascun passo è composto di vari task che cambiano di volta in volta, anche a seconda di eventuali interazioni del DSDM con altre metodologie e del tipo di situazione/progetto.

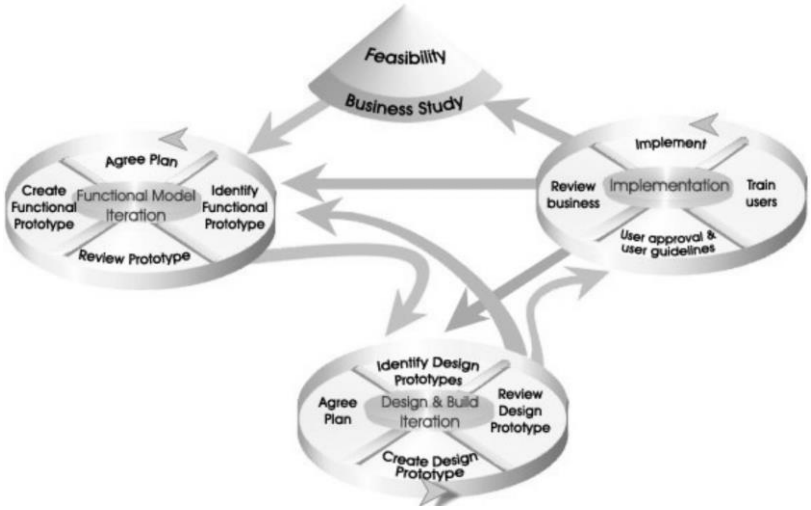

Figura 3.5: *Processo DSDM*

Come si vede in Figura 3.5 le 7 fasi di cui il DSDM è composto sono:

1) **Pre-project:** Durante questa fase viene deciso se e come il progetto deve essere fatto tra i vari suggeriti.

2) **Studio di fattibilità:** In questo step viene definito il raggio d'azione e vengono valutate la fattibilità economica, in termini di costi, e quella tecnica, in termini di possibilità di sviluppo del sistema.

3) **Studio lato business:** Vengono prese ulteriori decisioni su come debba essere organizzato il lavoro successivo, vengono chiariti maggiormente alcuni aspetti e raccolti i requisiti.

4) **Functional Model Iteration (FMI):** Durante questa fase vengono raffinati gli aspetti individuati nella fase precedente.

5) **Design and Build Iteration (DBI):** Questa fase permette di progettare e realizzare il prodotto in modo che sia consegnabile al cliente.

6) **Implementation:** Grazie a questo step si passa dall'ambiente di sviluppo a quello operativo.

7) **Post- project:** Questa fase tipicamente avviene 6 mesi dopo la conclusione del progetto evengono fatte le misure del caso e vengono valutati possibili miglioramenti.

Come è visibile in Figura 3.5 FMI, DBI e l'implementazione sono presentate come iterazioni, poiché devono essere svolte durante ciascun incremento. Oltre a queste fasi, insieme a ciascun task, devono essere attributi dei risultati concreti che devono essere raggiunti di volta in volta e che verranno opportunamente misurati.

Il DSDM oltre ai 7 step, prevede 4 tecniche abilitanti:

- L'utilizzo di timebox al posto delle milestone. Questi non sono altro che intervalli temporali di breve durata (solitamente 6 settimane) in cui devono essere svolti un certo numero di task.

- Le regole di MosCow che servono a prioritizzare le varie feature, in base a ciò che serve maggiormente al cliente in ciascun momento. I bisogni dei clienti infatti tendono a cambiare nel momento in cui individui nuove possibilità tecniche o in cambi l'ambiente di riferimento. Per questo è necessario sapere cambiare velocemente le funzionalità e farlo seguendo una certa classificazione logica. Le regole di MosCow prevedono di ordinare le feature in 4 classi sequenziali: Must Have, Should Have, Could Have e Want to Have.
- L'utilizzo di un prototipo che permette un rilascio frequente e incrementale. Questi permettono inoltre di osservare anticipatamente eventuali criticità e ottenere, fin dalle prime fasi, feedback degli utenti.
- L'utilizzo di "*Whorkshop*" in modo da facilitare la collaborazione tra utenti e sviluppatori. Durante tutti gli step del processo di sviluppo, tranne che nel pre e post- project, devono essere organizzati questi incontri durante i quali deve esserci la partecipazione attiva di tutte le persone coinvolte e in cui deve essere individuata una terza parte indipendente, che funga da moderatore tra sviluppatori e utenti.

3.6 Conclusione

Dopo aver visto i motivi che hanno portato alcuni esperti a creare nuovi metodi di sviluppo del software, i quali sono stati poi esportati in numerosi altri ambiti, e aver fatto un excursus di alcuni dei più importanti tra questi, verrà affrontato nel prossimo capitolo un caso di studio reale. In seguito infatti non solo verrà mostrato cosa vuol dire utilizzare un metodo Agile, soffermandosi maggiormente sulla fase di raccolta dei requisiti e lo sviluppo di un prototipo ad hoc, ma si cercherà di capire cosa accadrebbe se per lo stesso progetto venisse usato il metodo waterfall. Infine si tenterà, attraverso la letteratura e le conoscenze di alcuni esperti del settore, di ricostruire in quali ambiti è più efficace l'utilizzo di metodi Agile e in quali, ancora oggi, è preferibile adottare metodi tradizionali.

Si riporta inoltre in seguito, per completezza di informazioni, la tabella precedentemente mostrata, con l'aggiunta della colonna relativa alle metodologie Agile:

Model / Features	Waterfall	Prototype	Incremental	Spiral	RAD	Agile
Requirement Specifications	Beginning	Frequently changed	Beginning	Beginning	Time-box released	Frequently changed
Understanding Requirements	Well Understood	Not well understood	well understood	Well understood	Easily understood	Well understood
Cost	Low	High	low	expensive	low	Very high
Guarantee of Success	Low	Good	High	High	Good	Very high
Resource Control	Yes	No	Yes	Yes	Yes	No
Cost Control	Yes	No	No	Yes	Yes	Yes
Simplicity	Simple	simple	Intermediate	Intermediate	very simple	Complex
Risk Involvement	High	low	Easily manage	Low	Very low	reduced
Expertise Required	High	Medium	High	high	Medium	Very high
Changes Incorporated	Difficult	Easy	Easy	Easy	Easy	Difficult
Risk Analysis	Only at beginning	No risk analysis	No risk analysis	Yes	Low	Yes
User Involvement	Only at beginning	High	Intermediate	High	Only at the beginning	High
Overlapping Phases	No such phase	Yes	No	Yes	No	Yes
Flexibility	Rigid	Highly flexible	Less flexible	Flexible	High	Highly flexible
Maintenance	Least glamorous	Routine *maintenance*	Promotes maintainability	Typical	Easily maintained	Promotes maintainability
Integrity & Security	vital	Weak	Robust	High	Vital	Demonstrable
Reusability	limited	Weak	Yes	Yes	some extent	Use Case *Reuse*
Interface	minimal	crucial	Crucial	Crucial	minimal	*model-*driven
Documentation &Training required	vital	weak	Yes	Yes	limited	Yes
Time Frame	Long	Short	Very long	Long	Short	least possible

Tabella 3.3

4 | UTILIZZO DELLA METODOLOGIA AGILE NELLO SVILUPPO DI UN NUOVO APPLICATIVO PER I CONTACT CENTER INBOUND

In questa sezione verrà descritta l'esperienza svolta durante lo stage presso la nota società di consulenza Accenture S.p.A., la quale si occupa di progetti in ambito direzionale, IT e outsourcing.

Il progetto al quale ho partecipato ha previsto lo sviluppo di un nuovo applicativo da installare sui computer dei Call Center inbound (centri di gestione delle problematiche post-vendita in cui il flusso della chiamata è dal cliente verso l'operatore) appartenenti ad una delle più grandi società di telecomunicazione che operano in Italia, utilizzando un approccio di tipo Agile.

Tale lavoro è stato svolto partendo da un innovativo sistema tecnologico di ricerca già esistente, che aggrega e correla velocemente i dati, proiettandoli in modo semplice su un'unica pagina sviluppata in HTML5. Questa pagina che appare sulla schermata dei device è composta da una serie di widget, contenenti le diverse informazioni, in modo tale che siano facilmente visualizzabili.

In particolare il mio incarico durante questi mesi ha previsto una serie di visite on site presso diversi Contact Center italiani, al fine di reperire i requisiti lato utente e supportare da lato business il team di sviluppo del prototipo. Questo mi ha permesso di osservare da vicino l'utilizzo della metodologia Agile per queste prime fasi e di capirne l'importanza in contesti di questo tipo.

4.1 Ambito di nascita del progetto e obiettivi

Nell'ambito delle società di telecomunicazioni il mercato di riferimento risulta ormai completamente saturo e la possibilità di intercettare nuovi clienti risulta pressoché nulla. Data questa realtà dei fatti, i giganti di questo settore possono solamente adottare strategie orientate a sottrarsi quote di mercato e a diminuire la possibilità di rischio churn, ovvero diminuire il numero di clienti che abbandonano la compagnia per passare ad uno dei competitor. Per perseguire questa strategia diventa importante la soddisfazione del cliente attraverso ogni sua forma, compreso anche il miglioramento della customer experience durante la gestione delle problematiche post-vendita. Quando la relazione tra brand e cliente è positiva, questa può portare ad un commitment che dura una vita, viceversa si incorre nel pericolo di abbandono da parte del cliente in favore di un competitor. Secondo il report del 2011 sul Customer Experience Impact, commissionato da RightNow, sono emersi i seguenti risultati:

- 86% dei clienti pagherebbe di più per una customer experience migliore
- 89% dei clienti passa ad un competitor quando percepisce una pessima customer experience

Ecco perché sempre secondo questa indagine, le aziende usano diverse tattiche per soddisfare ciò che i clienti vogliono:

- Utenti di interfaccia amichevoli
- Possibilità di avere in tempi veloci le informazioni di cui hanno bisogno
- Buona reputazione del brand

In quest'ottica è nata l'esigenza di migliorare la qualità percepita dal Cliente che chiama i Call Center dell'azienda a causa di problematiche relative al proprio telefono, per ottenere informazioni su promozioni e offerte o per chiarimenti sulle movimentazioni di credito e traffico dati. Per conseguire l'obiettivo, si è

avvertita la necessità di andare a migliorare i sistemi presenti sui device degli operatori telefonici e passare dall'avere numerosi applicativi, che aprono decine di finestre sullo schermo del computer, ad un unico sistema principale di interfaccia con un'unica landing page intuitiva dalla quale reperire le informazioni precedentemente aggregate. Questo nuovo sistema basa il suo funzionamento su un motore di ricerca semantica e sulla contestualizzazione dinamica della pagina principale, formata da numerosi widget che si popolano in base alla tipologia del cliente e della chiamata, evitando così all'operatore di dover aprire diversi sistemi, causando un rallentamento nella gestione della telefonata.

Si riporta in Figura 4.1 una schematizzazione del passaggio dalla situazione as-is a quella to-be.

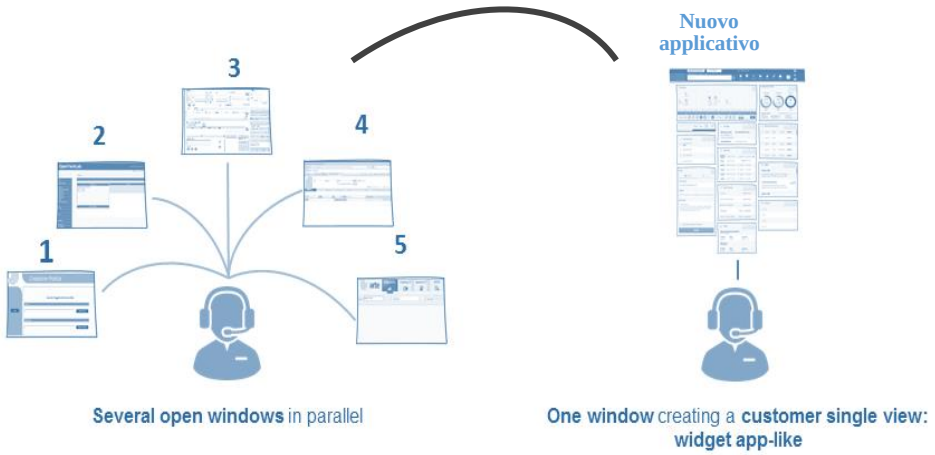

Figura 4.1: *Passaggio da N viste ad una sola*

Ad oggi infatti, ogni qual volta un operatore telefonico riceve una telefonata da parte di un Cliente deve aprire un ingente numero di sistemi in base alle richieste e alle problematiche della persona chiamante.

Si prenda ad esempio un caso banale come quello in cui il Cliente chiama perché non si ritrova con il credito residuo sul proprio cellulare, teoricamente risolvibile in pochi passaggi.

Nella situazione as-is per reperire tutte le informazioni l'operatore deve:

1) Verificare il credito residuo e lo storico delle ricariche in un certo periodo di tempo sul Sistema 1
2) Verificare eventuali addebiti dovuti al traffico voce, dati ed sms sul Sistema 2
3) Verificare se sul cellulare sono attivi Servivi a Pagamento sul Sistema 3
4) Verificare l'eventuale presenza di promozioni non richieste dal Cliente nuovamente sul Sistema 2
5) Riaccreditare l'importo scalato al Cliente, se sussistono le condizioni, sul Sistema 4
6) Compilare i campi, con le opzioni predefinite, in modo da tracciare il Motivo della Chiamata sul Sistema 5

In questo caso sono stati aperti 5 sistemi diversi, dove su ciascuno, per reperire le informazioni utili, devono essere aperte ulteriori finestre, che non sono visibili a prima vista sulla schermata principale.

Si comprende come una situazione di questo tipo diminuisca le prestazioni dell'operatore, che è costretto a lavorare con una quantità elevata di informazioni, contenute su finestre diverse che devono essere aperte e richiuse in modo continuo sullo schermo del pc durante la telefonata. Tutto ciò va ad impattare sulle tempistiche della telefonata. Queste si allungano inevitabilmente per consentire all'operatore di reperire tutte le informazioni necessarie situate su sistemi diversi, provocando un'inefficienza nella gestione del Cliente. La ricerca infatti avviene con il Cliente in linea, il quale è così soggetto a lunghi tempi di attesa o all'inefficacia delle risposte ricevute.

Data la complessità nel reperire le informazioni, l'operatore, se non grazie ad una notevole esperienza, rischia di tralasciare aspetti fondamentali per tutti i lavoratori che trattano direttamente con la clientela. Agli operatori di front-line sono infatti richieste caratteristiche quali la cordialità e la disponibilità ad assecondare qualsiasi esigenza del Cliente.

Il lavoro dell'agente di Call Center si trasforma quindi in un vero e proprio trade-off tra velocità e abilità nella ricerca e gestione della persona chiamante.

Per questo motivo è stato scelto di sviluppare un applicativo in HTML5 che fosse formato da un'unica pagina principale, visualizzabile sul pc dell'operatore e che riportasse tutte le informazioni utili durante la gestione della chiamata, allocate al momento su più sistemi diversi.

La pagina presentata agli addetti nei Call Center non solo mostra direttamente, su un'unica schermata, tutte le informazioni, ma le raggruppa in base a cluster di appartenenza su appositi widget. Ad esempio sarà presente un widget dedicato alle offerte, riportante le promozioni attive sul numero del Cliente, e un widget dedicato alle movimentazioni avvenute sullo specifico numero. In questo modo l'operatore potrà recuperare i dati cercati, direttamente sulla schermata principale del computer o al più massimizzando uno specifico widget per reperire ulteriori dettagli.

L'applicativo inoltre si popola in maniera intelligente, prendendo come input il numero chiamante. Questo consente di mostrare sulla pagina tutti i dati relativi al Cliente associato a quel numero. A seguito del caricamento dei dati il sistema permette di effettuare ricerche grazie all'utilizzo di parole chiave e di ricontestualizzare la pagina qual ora il Cliente avesse problemi su di una linea diversa da quella chiamante. Questa funzionalità può essere abilitata semplicemente cliccando sul numero desiderato, presente in un widget ad hoc contenente tutte le linee associate al Cliente.

Una situazione di questo tipo permette all'addetto del CC di non navigare tra sistemi diversi e di reperire le informazioni utili in tempi minori, accorciando le tempistiche di gestione della problematica.

Questa facilitazione nel lavoro di ricerca permette di agevolare l'altro lavoro dell'operatore: la gestione del Cliente durante la telefonata. L'addetto infatti può adoperare le sue skill per fornire un'esperienza piacevole e soddisfacente alla clientela chiamante, aspetto fondamentale nei servizi post-vendita.

Infine la facilità di utilizzo del nuovo applicativo permette di aumentare le performance degli operatori a parità di training iniziale, permettendo così una diminuzione dello stress e dell'insoddisfazione tipiche di questo lavoro.

Durante il business case, ovvero nella fase di studio preliminare che ha comparato la soluzione as-is con quella to-be sono stati presi come driver alcuni KPI. Questi sono stati scelti per misurare l'impatto positivo della nuova soluzione sulle performance degli operatori e l'efficienza del Call Center.

Secondo due studi condotti da Feinberg e Cheong, riportati nel libro "10 KPIs That Affect Customer Satisfaction with Call Center Service" condotti su 512 Call Center, i migliori KPI che impattano sulla soddisfazione del Cliente sono i seguenti:

1) One Call Resolution: si tratta della percentuale di chiamate che l'operatore riesce a risolvere senza dover trasferire la chiamata o richiamare il Cliente
2) Average time queue: tempo medio di attesa in coda del Cliente, quando prova a chiamare un Call Center
3) Average After Call Work Time: Tempo medio di lavoro speso dall'operatore per finire di completare il lavoro, dopo che ha chiuso la chiamata
4) Service level: numero di chiamate gestite in un certo intervallo di tempo
5) Agent turnover rate: percentuale di end-user che abbandonano il lavoro nei Call Center
6) Average Speed of Answer: tempo medio di risposta da parte dell'operatore
7) Average handling time: tempo medio di gestione della chiamata, compresi sia i momenti in cui l'operatore interagisce col Cliente che i momenti di attesa e il tempo che serve all'operatore per finire il lavoro una volta chiusa la chiamata

8) Average abandonment rate: percentuale di Clienti che chiudono la chiamata prima di parlare con l'operatore
9) Percentage of Calls Blocked: percentuale di Clienti che ascoltano la suoneria quando chiamano il CC
10) Schedule Adherence: misura del grado di conformità ai programmi assegnati

A sostegno del business case sono stati usati alcuni di questi KPI, la cui variazione deve consentire un payback dell'investimento iniziale in tempi brevi. E' stato stimato infatti che la decisione di investimento su questo nuovo sistema dovesse essere remunerata con un timing di circa 17 mesi. Dunque per misurare la bontà di questa soluzione sono stati scelti i seguenti indicatori:

- **One Call Resolution (OCR)**: indica le chiamate che riescono ad essere gestite in un'unica volta, senza la necessità di ricontatto del Cliente o di trasferimento della telefonata. Questo KPI dovrebbe tenere traccia della capacità del nuovo applicativo di facilitare la ricerca delle informazioni da parte dell'end-user e offrire un esperienza migliore al Cliente.

  Si tratta di un indicatore fondamentale, che se costantemente misurato e analizzato porta ad una lunga serie di benefici come:

  - Aumento della soddisfazione del Cliente: un incremento di questo indicatore migliora la soddisfazione e diminuisce il rischio che il Cliente passi ad un altro operatore. Questo è dovuto al fatto che i consumatori vogliono parlare con addetti competenti, che risolvano in tempi brevi i loro problemi.
  - Riduzione dei "Consumatori a rischio": secondo uno studio infatti solo il 2% dei Clienti che ha risolto la problematica in una chiamata cambierebbe operatore. Viceversa il 19% dei Clienti a cui non viene data una soluzione all'interno della chiamata, cambierebbero operatore.
  - Riduzione dei costi operativi: Nel caso di questo progetto è stato scelto questo KPI perché grazie al suo incremento i CC in-house

dell'azienda sono in grado di gestire un volume maggiore di telefonate. Attualmente infatti il numero di chiamate gestite dai CC di proprietà dell'azienda è del 30% mentre i volumi di chiamate trasferite agli outsourcer è circa del 70%.

Grazie ad un incremento previsto dell'1% di questo indicatore i CC riuscirebbero a gestire il 30% in più delle telefonate, permettendo un notevole risparmio di costi.

- Miglioramento delle performance e della soddisfazione dell'operatore: Grazie al nuovo applicativo dovrebbe aumentare l'OCR, dato che grazie alla sua semplicità di utilizzo le performance e la soddisfazione dell'end-user migliorano notevolmente. Secondo uno studio nel caso di un alto OCR gli operatori si sentono meno stressati e più soddisfatti del loro lavoro, apportando sicuramente un beneficio per l'azienda.

- **Average Handling Time**: rappresenta il tempo medio di gestione della chiamata, comprensivo anche dei momenti di attesa e di lavoro una volta che il Cliente ha chiuso la telefonata. Esso è calcolato come:

$$\frac{Total\ talk\ time + Total\ Hold\ Time + Total\ Wrapup\ Time}{Number\ of\ Calls\ Handled}$$

All'interno di questo indicatore viene incluso il *Total Hold Time,* poiché altrimenti l'operatore potrebbe passare più tempo in attesa in modo da ridurre l'AHT. Un discorso equivalente è valido anche per il Total Wrapup Time, che potrebbe altrimenti essere allungato dagli operatori per svolgere delle pause prima di prendere la chiamata successiva.

Il fatto che questo KPI sia alto potrebbe essere dovuto da una serie di fattori quali:

- Sistemi del computer lenti
- Scarso addestramento dell'utente

- Frequente passaggio da un sistema ad un altro
- Lunghi tempi di attesa

Il nuovo sistema, presentando tutte le informazioni su di un'unica schermata riduce i tempi necessari per passare da un sistema all'atro.
Inoltre grazie al nuovo applicativo l'operatore dovrebbe essere in grado di ricercare i dati che portano alla risoluzione del problema dell'operatore in modo più veloce, diminuendo i tempi morti della chiamata.
Infine la facilità di utilizzo del nuovo sistema, dovrebbe aumentare le performance a parità di addestramento. L'interfaccia che si presenta all'utente è notevolmente intuitiva e consente di ridurre le ore di training, consentendo di risparmiare i relativi costi.

- **Produttività media agente (PMA)**: in termini di numero di chiamate gestite dall'operatore. Questo indicatore è riconducibile al Service Level riportato negli studi citati sopra. Grazie al nuovo applicativo, alla rapidità di ricerca delle informazioni e alla facilità di utilizzo, gli end-user dovrebbero essere in grado di aumentare il numero di chiamate gestite in un certo intervallo di tempo. L'aumento di questo indicatore dovrebbe consentire di ridurre il numero di commesse affidate attualmente agli outsourcer, favorendo la gestione delle chiamate in house e aumentando i savings. Questo è possibile grazie alla diminuzione del numero di chiamate passate a centri terzi di gestione del Cliente, che rappresentano un costo variabile per l'azienda. A differenza dei Contact Center in house, che rappresentano un costo fisso, gli outsourcer presentano una variabilità nei costi, dovuta al numero di chiamate gestite.

Oltre a questi indicatori tipici dei Call Center, è stato utilizzato un KPI fondamentale per la valutazione di investimenti in nuove tecnologie:

- **Total cost of ownership:** Si tratta dei costi di acquisizione del sistema sommato ai costi di manutenzione dello stesso, all'interno di tutto il ciclo di vita della tecnologia. Un'analisi di questi costi deve essere svolta ogni qual volta debba essere fatto un investimento di capitale. Tra i costi presi a riferimento vengono inclusi sia quelli relativi all'hardware e al software, che quelli relativi all'addestramento, alle licenze, ai possibili rischi futuri (e.g: disponibilità di aggiornamenti del sistema, licenze, rottura del sistema), spese per gli aggiornamenti o per il rimpiazzo del sistema ecc.

  In particolare nel confronto tra una tecnologia esistente ed una nuova soluzione dovrebbe essere tenuto conto dei costi di manutenzione. Nel caso del progetto è stata stimata una riduzione di questo indicatore in quanto sviluppare, implementare e manutenere il nuovo applicativo risulta meno oneroso rispetto alla manutenzione dei vecchi sistemi.

Infine è stata scelta la seguente metrica per recepire l'impatto del nuovo sistema sul Cliente finale:

- **Net Promoter Score (NPS)**: è un parametro che misura la fedeltà del Cliente al brand, ormai diventato l'indicatore principale nell'ambito delle aziende di telecomunicazione. Esso misura il grado di soddisfazione del consumatore in base a quanto consiglierebbe un prodotto/servizio ai propri amici, parenti o colleghi. La scala di riferimento, secondo cui il cliente consiglierebbe il prodotto, comprende i valori da 0 a 10 e si divide in:
  - Promotori: coloro che rispondo da 9 a 10
  - Passivi: coloro che rispondo da 7 a 8
  - Detrattori: coloro che rispondo da 0 a 6

  In seguito a questa categorizzazione l'NPS viene calcolato sottraendo alla percentuale dei promotori quella dei detrattori.

Tale indicatore viene utilizzato per valutare il lavoro degli operatori dei Call Center da parte dei Clienti al termine di ogni chiamata avvenuta.

La possibilità di miglioramento di questi indicatori comporta come fattore ultimo la soddisfazione del Cliente. Viene infatti diminuito il tempo di attesa da parte del Cliente per la risoluzione del motivo di chiamata e aumentata la probabilità di risoluzione della problematica con un unico intervento. Inoltre l'applicativo, agevolando il lavoro dell'operatore, dovrebbe permettergli di concentrarsi di più su aspetti di interazione col cliente. Questo dovrebbe apportare come risultato una maggiore cordialità e disponibilità nella gestione della telefonata.

Questo innesca nel Cliente un meccanismo di fiducia nei confronti della compagnia, diminuendo la possibilità di abbandono per un brand concorrente. Egli si sente seguito e ascoltato, anche durante le fasi successive all'acquisto del prodotto e la capacità mostrata dall'azienda di sapere rispondere prontamente alle sue richieste, fortifica il rapporto di fidelizzazione, che era stato messo alla prova dal presentarsi della problematica.

In Tabella 4.1 si riporta in termini percentuali, il miglioramento previsto di alcuni di questi indicatori, dovuta all'implementazione del nuovo sistema:

KPI	Stima %
One Call Resolution	+ 1%
Average Handling Time	-25%
Produttività Media Agente	+ 8,5%

Tabella 4.1: Stima di miglioramento dei principali driver di progetto

Oltre alle stime numeriche si è intravisto, attraverso questo progetto, la possibilità di migliorare l'NPS, consentendo all'operatore di dedicare più tempo

ad attività a valore aggiunto e di ridurre il tempo impiegato per addestrare un nuovo operatore.

Per portare avanti questo progetto, conformemente alla tendenza sviluppata negli ultimi anni e perseguita in primis da Accenture nell'ambito di molti progetti, è stato scelto di adoperare la metodologia Agile.
Data la complessità del lavoro, che prevede di eliminare i vecchi sistemi, e la decisione di sviluppare il sistema in 3 release, si è vista la necessità di adottare un approccio rapido e snello.
Proprio nella fase precaria di raccolta dei requisiti è stato deciso di usare un approccio Agile. L'intento è quello di ridurre le tempistiche e consentire agli operatori dei CC di esprimere i propri bisogni, richieste, dubbi e perplessità al fine di sviluppare un applicativo con delle funzionalità migliori di quelle as-is.

Infatti il grande numero di requisiti che devono essere raccolti e la necessità di ascoltare, non solo la voce del dell'azienda Cliente, ma anche la voce dell'end-user (l'operatore dei Call Center), hanno fatto sì che la scelta ricadesse sull'utilizzo di una metodologia Agile, piuttosto che su un metodo tradizionale.

In questo contesto, in cui il nuovo applicativo andrà a sostituire i sistemi precedentemente usati da un cospicuo numero di operatori, risulta fondamentale capire il modo in cui vengono utilizzati gli applicativi presenti e comprendere i bisogni, espressi e latenti, degli operatori. Tutto questo viene fatto nell'ottica di sviluppare la migliore soluzione possibile e cercare di ridurre gli inevitabili disagi dovuti al cambiamento nel modo di lavorare.
Oltre al miglioramento della User experience, dove User è l'operatore del Call Center che percepisce in modo personale l'esperienza tra se stesso e il nuovo prodotto, la voce dell'operatore svolge anche la funzione di migliorare la Customer experience.

Il contatto sviluppato durante le telefonate, permette all'end-user di percepire le volontà del Cliente e di farsi portavoce delle sue esigenze, durante la fase di raccolta dei requisiti.
La conoscenza delle esigenze del Cliente finale viene così utilizzata per sviluppare un applicativo che vada a migliorare anche l'esperienza del vero e proprio Customer.

Infine la necessità di attuare un Transformation vera e propria, la quale richiede di partire da zero, eliminando i sistemi precedenti e il fatto di avere già una soluzione esistente intorno alla quale devono essere costruiti i requisiti, hanno confermato la scelta di questo approccio.

4.2 Struttura del progetto

All'interno del questo paragrafo verrà descritto in che modo è stato strutturato il lavoro, in modo da riuscire a mappare propriamente i sistemi presenti attualmente nei Call Center ed andare a recepire i requisiti di tutte le parti interessate, cercando di mettere in luce gli aspetti che hanno reso Agile questa parte del progetto.

L'obiettivo strutturale del progetto prevede lo sviluppo dell'applicativo in 3 release, così composte:

- Release 1: Sviluppo dell'applicativo in ambito Consumer, dotando il sistema delle capacità di tipo informativo
- Release 2: Sviluppo dell'applicativo in ambito Consumer, dotando il sistema delle capacità di tipo dispositivo
- Release 3: Sviluppo dell'applicativo in ambito Enterprise

Le quali dovranno essere svolte nelle tempistiche riportate nel Gantt in Figura 4.2:

Figura 4.2: *Gantt del progetto*

In particolare la fase iniziale della Release 1 prevede che lo svolgimento delle seguenti fasi sia completato prima della data di Approvazione del resto del progetto da parte dell'azienda Cliente:

- *Pianificazione delle attività*: in termini di tempi, risorse, attività da svolgere, fonti da cui reperire i requisiti e indicatori da utilizzare
- *Solution definition*: scelta degli use-case da analizzare e delle funzionalità di base che dovranno essere già presenti sull'applicativo, definite insieme all'area business dell'azienda per cui viene sviluppato l'applicativo
- Processo di:
  - *Raccolta dei requisiti*: attività che prevede di reperire presso alcuni end-user selezionati i requisiti da includere nello sviluppo dell'applicativo
  - *Design degli Use-Case*: attività svolta per comprendere in che modo vengono gestite sui sistemi as-is le varie tipologie di chiamate ricevute dai CC, sulla base degli UC individuati
  - *Prototipazione*: attività che serve a mostrare all'area business e agli operatori dei CC i risultati ottenuti, in modo tale da poter raccogliere eventuali feedback

4.2.1 Pianificazione delle attività

Durante questa fase iniziale sono state pianificate le attività da svolgere durante il progetto. In particolare si riporta la pianificazione delle attività previste per i 3 mesi successivi all'inizio del progetto, prima della data di approvazione del progetto da parte della società Cliente. Per questa prima fase è stata individuata la necessità di svolgere le attività di raccolta dei requisiti, mappatura degli Use-Case e sviluppo del prototipo direttamente presso alcuni Call Center.

Come si è visto nel capitolo precedente infatti, all'interno di un progetto che utilizza l'approccio Agile è fondamentale la collaborazione tra tutte le parti interessate, ovvero tra il team addetto al business e allo sviluppo della soluzione, il Cliente e gli utilizzatori finali dell'applicativo.

Questa collaborazione è tanto più fondamentale nella parte di raccolta dei requisiti, durante la quale tutti questi stakeholder devono essere in grado di esprimere le proprie richieste, osservazioni e suggerimenti.

L'importanza data a questa fase dovrebbe permettere di sviluppare una soluzione migliore, in tempi minori.

Per questo motivo è stato deciso di raccogliere i requisiti da 2 fonti principali:

- Headquarter della società di telecomunicazione sopra citata
- End-User dei Call Center di questa società

Per quanto riguarda l'Headquarters i requisiti sono giunti sia dal lato business che tecnico, in modo tale da non perdere aspetti relativi a ciascuna delle due aree, e sono stati raccolti in un documento apposito, che può essere ricondotto al Product Backlog visto nello Scrum.

Questa attività è iniziata durante la fase di definizione della soluzione, in modo tale da utilizzare quanto raccolto per mostrare agli end-user presso i Call Center una versione base dell'applicativo. Il coinvolgimento dell'HQ è stato presente anche nelle fasi successive per integrare i requisiti raccolti on-site e per supervisionare le funzionalità

da implementare, in modo tale da essere coerenti con le policy aziendali di gestione degli operatori e di sicurezza dei dati.

La vera novità però, che differenzia l'Agile dagli approcci tradizionali, è il coinvolgimento degli utilizzatori finali del sistema. Per avere la minor perdita possibile di informazioni relative alle funzionalità presenti sulla soluzione as-is rispetto a quelle che andranno implementate sulla soluzione to-be sono state selezionate alcune tipologie di End-User, specializzate in diversi ambiti. Ciascuna di queste categorie è stata individuata a seconda dello specifico CC di appartenenza e si è fatta rappresentante delle modalità di gestione di alcune tipologie di problematiche.

In particolare la struttura dei CC di questa azienda, come è visibile in Figura 4.3 è organizzata a stella, ovvero esistono 7 Call Center in-house, di proprietà dell'azienda intorno ai quali si collocano i Contact Center degli outsourcer.

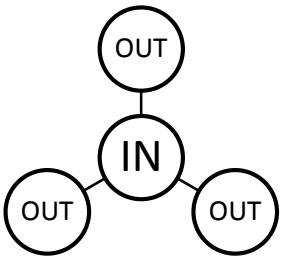

Figura 4.3: *Struttura a stella dei CC*

Grazie a questa struttura, il CC in-house trasferisce il volume di chiamate che non riesce a gestire giornalmente agli outsourcer di competenza. Ogni CC IN infatti ha dei Call Center OUT associati, che gli "gravitano" intorno, a cui girare le chiamate in eccesso.

Per lo scopo di questo progetto è stato scelto di raccogliere i requisiti degli end-user e andare a mappare i processi as-is che avvengono durante le diverse tipologie di chiamata solamente dei 7 CC in-house. Di questi 7 sono stati scartati i 3 che non si occupano della gestione della problematica del Cliente Consumer direttamente

durante la chiamata stessa. Infatti sia durante la Release 1 che la Release 2 il focus è incentrato sul mondo Consumer e solo in una fase successiva all'approvazione è stato previsto il recepimento dei requisiti per il mondo Enterprise.

In particolare in Tabella 4.2 vengono riportati i quattro Call Center scelti e una breve descrizione dell'area di cui si occupano:

#	Call Center	Tipo	Descrizione
1	Pisa	Consumer Mobile Prepaid	Call Center che gestisce chiamate di Clienti possessori di telefonia Mobile di tipo prepagato (con pagamento anticipato). Il focus delle attività è su chiamate di tipo informativo e di consulenza commerciale
2	Bologna	Consumer Mobile Prepaid Smart&New	Call Center che gestisce chiamate di Clienti possessori di telefonia Mobile di tipo prepagato (con pagamento anticipato). Il focus delle attività è su chiamate di Clienti nuovi e su problematiche di tipo tecnico
3	Catania	Consumer Mobile Postpaid	Call Center che gestisce chiamate di Clienti possessori di telefonia Mobile di tipo postpagato (con pagamento posticipato). Il focus delle attività è su chiamate di clienti abbonati e su problematiche di tipo tecnico
4	Ivrea	Consumer Residential	Call Center che gestisce chiamate di Clienti di Rete Fissa (telefono fisso, ADSL e fibra). Il focus delle attività è su chiamate di tipo informativo

Tabella 4.2: Quattro tipologie di Call Center

Durante le visite on-site è stato previsto di svolgere le attività precedentemente riportate:

- Mappatura degli Use Case individuati
- Raccolta dei requisiti
- Sviluppo del prototipo

La pianificazione temporale prevista per lo svolgimento di queste attività e per la fase di pianificazione stessa e di definizione della soluzione è riportata nel Gantt mostrato in Figura 4.4:

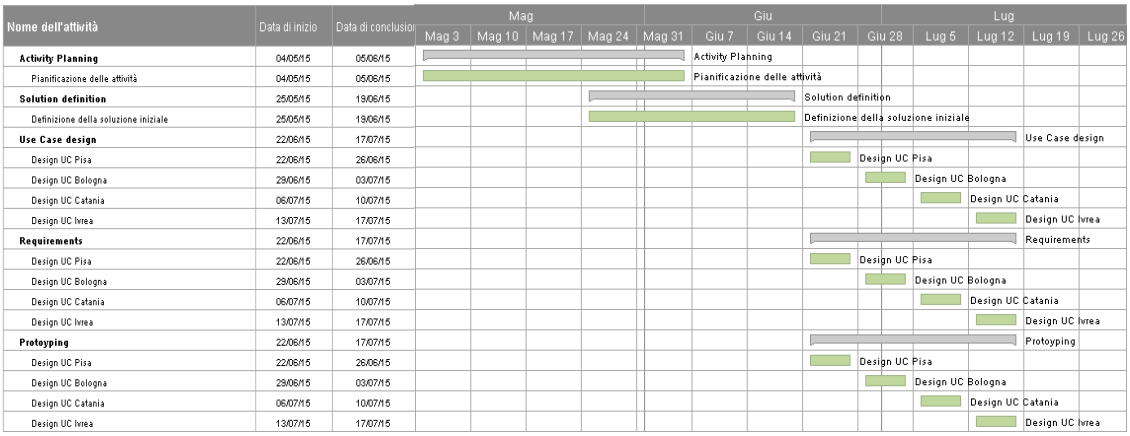

Figura 4.4: *Gantt del progetto*

Per la fase di solution definition sono state previste 3 persone lato Accenture che lavorassero insieme all'HQ dell'azienda Cliente per contribuire a definire le funzionalità di base dell'applicativo.

Infine per lo svolgimento delle attività di visite on-site è stato previsto l'utilizzo da parte di Accenture di un team composto da 8 persone che avessero capacità in ambiti differenti. La descrizione dei ruoli coperti da queste risorse verrà riportata successivamente, durante la fase di descrizione delle attività svolte presso i Call Center.

4.2.2 Solution definition

Durante questa fase sono stati raccolti in un apposito documento i requisiti funzionali di base che devono essere presenti sull'applicativo. Questa fase non solo recepisce le richieste derivanti dall'azienda Cliente, ma definisce anche una soluzione da mostrare come prototipo durante la visita presso il primo Call Center.

Inoltre per ciascuno di questi centri di gestione della chiamata sono stati individuati degli Use Case di riferimento in modo tale da ottenere la mappatura delle attività, e avere un confronto concreto tra la situazione as-is e quella to-be. Queste casistiche dovrebbero rappresentare situazioni reali che si trovano a gestire i Rep (Call Center Representative) durante il momento di interazione con Cliente.

La decisione su quali Use Case adoperare ha visto la scelta tra due diverse modalità di mappatura:

1) Utilizzare i Motivi di Chiamata: si tratta di insiemi, aggregati per tipologia, dei Motivi di Contatto, ovvero dei riferimenti attraverso cui tracciare la chiamata e l'attività dell'operatore. Al termine di ogni chiamata infatti l'end-user deve riportare le motivazioni della chiamata da parte del Cliente, andando a selezionare 3 diversi stati predefiniti, che identificano in maniera univoca il contatto avvenuto tra Cliente e Call Center. Ad oggi i Motivi di Chiamata sono 12 e clusterizzano al loro interno 96 diversi Motivi di Contatto
2) Utilizzare le procedure seguite dagli operatori, durante la gestione della chiamata: gli end-user infatti utilizzano un catalogo dell'azienda Cliente, sul quale sono presenti 146 procedure da seguire a seconda del problema della chiamata. Per ognuna di queste sono riportati diversi step, con domande specifiche a risposta chiusa, da sottoporre al Cliente. Tali procedure hanno una struttura ad albero, percorribile in una sola direzione, in cui ad ogni nodo corrisponde una domanda. Queste procedure servono per far sì che l'operatore segua un percorso ben preciso nella gestione della chiamata e non prenda liberamente decisioni su possibili soluzioni da proporre al Cliente.

Dopo un'attenta analisi dei due approcci la scelta è ricaduta sulla seconda modalità, poiché nonostante la possibilità di utilizzare tutti e 12 i Motivi di Chiamata questi non sarebbero stati facilmente confrontabili con le azioni svolte dagli operatori nello stato as-is dei sistemi.

Utilizzando il secondo caso si è riusciti a mappare con 97 procedure, ovvero il 70% del totale, tutte le azioni svolte sui sistemi odierni. Il 30% trascurato infatti è stato coperto ugualmente in termini di funzionalità utilizzate sui sistemi, non comportando quindi alcuna perdita informativa.

In conclusione quindi sono stati individuati 97 UC, ripartiti sui 4 diversi CC, come dettagliato in Tabella 4.3:

Call Center	N° Use Case	Esempio
Pisa	23	Non mi torna il credito
Bologna	19	Gestione contestazioni roaming
Catania	19	Problemi amministrativi
Ivrea	35	Fattura ADSL/Fibra

Tabella 4.3: Use Case individuati

Questi Use Case, divisi per CC, sono stati a loro volta clusterizzati per macro aree di appartenenza ed è stata individuata per ciascuno la frequenza di accadimento. Questa si distingue in alta, media o bassa in base al numero di chiamate ricevute per quella problematica.

Sulla base di queste casistiche, è stato predisposto un documento per raccogliere in una fase successiva le informazioni utili a mappare i sistemi presenti e capire in che

modo gli Operatori lavorano. Inoltre sono stati individuati i KPI da rilevare durante le visite on-site per confrontare la situazione as-is con quella to-be all'interno dei CC.

L'obiettivo di questi KPI è quello di dimostrare l'efficienza della nuova soluzione rispetto alla precedente in termini di velocità di gestione della chiamata e di user experience dell'operatore.

In particolare per ciascun UC il documento deve riportare le seguenti informazioni:

- *Customer need*: si tratta del bisogno manifestato dal cliente, che lo spinge a contattare il Contact Center
- *Attività del Rep:* sono le attività che l'operatore svolge sui sistemi as-is mentre è in cuffia con il cliente
- *Info/Dati necessari*: informazioni indispensabili per svolgere ciascuna attività individuata al punto precedente
- *Sistemi sorgente dati*: per ciascuna attività il Rep deve solitamente accedere ad un sistema diverso. In questa sezione sono segnati i nomi degli applicativi utilizzati durante tutti gli step di gestione della chiamata
- *KPI*:
    - Numero di click: click del mouse effettuati dall'operatore per svolgere un'attività di verifica informativa o di azione dispositiva all'interno di uno specifico Use Case. Nella situazione as-is avendo le informazioni sparse devono essere utilizzati più click, rispetto al to-be, dove le informazioni sono tutte concentrate su di un unico applicativo.
    - Numero di schermate: finestre aperte sullo schermo del computer dell'operatore durante ciascuna attività. In questo caso dovrebbe essere mostrata la netta differenza tra la situazione as-is in cui si hanno più sistemi aperti, ognuno riportante le informazioni su più finestre, e la situazione to-be in cui si ha un'unica landing page, dove al massimo vengono massimizzati dei widget.
    - AHT (Average Handling Time): tempo medio di gestione di una chiamata riferita ad uno specifico Use Case. Ovvero tempo che l'end

user adopera per effettuare tutte le attività in modo tale da fornire una soluzione al Cliente. Questo dovrebbe riportare un notevole miglioramento grazie alla minore dispersione delle informazioni e la facilità di ricerca da parte dell'end-user nella situazione to-be.

Questi KPI sono stati utilizzati in un momento successivo alla loro raccolta per confrontare la soluzione as-is con quella to-be. Grazie all'utilizzo del prototipo è stato possibile infatti fare un paragone tra i sistemi vecchi e quello nuovo, in modo da mostrare concretamente i vantaggi ottenibili nel secondo caso.

- *Pain Points*: maggiori svantaggi dovuti dallo svolgere l'attività nello stato as-is. Si tratta delle problematiche e difficoltà riscontrate durante l'utilizzo dei sistemi presenti oggi

Questo documento di raccolta, così formato, è stato utilizzato come guida per rapportarsi con gli end-user. Questo ha permesso successivamente di focalizzarsi sulle azoni realmente svolte sui sistemi, senza rischiare di tralasciare aspetti fondamentali durante le visite ai CC. Se ne mostrano dei frammenti in Figura 4.5:

						KPIs			
Frequenza	Use case list	Customer need	Attività del rep	Info / dati necessari	Sistemi sorgente dati	# clicks	# schermate	AHT	Pain points
ALTA	Non mi torna il credito		Verifica credito residuo in	Credito residuo e storico ricariche sulla SIM chiamata / indicata dal cliente		9	4	8 min	
			Verifica dettaglio chiamate / traffico generato (addebiti, da quando non torna il credito, ecc...)	Dettaglio traffico (voce e dati)		3	1		
				Dettaglio traffico (SMS/MMS)		3	1		
				Dettaglio traffico (VAS)		7	5		
			Verifica servizi a pagamento	Numero da cui cliente riceve messaggi a pagamento		4	1		
			Verifica esistenza promo attive non richieste dal cliente	Consistenza servizi attivi sulla SIM		5	4		
			Inserimento Motivo di contatto (MdC)			10	2		
						12	3		
						41	18		

Figura 4.5: *Documento di raccolta dati sullo stato as-is*

4.2.3 Struttura del processo di raccolta dei requisiti, design degli use-case e prototipazione

Il processo di raccolta dei requisiti e sviluppo del prototipo è stato strutturato in maniera Agile, prevedendo un approccio iterativo e incrementale.

Il progetto è stato scomposto in 4 iterazioni, che si ripetono ciclicamente e il cui risultato funge da base per la fase successiva. Ciascuna iterazione, sempre secondo l'approccio Agile, deve essere iniziata e portata a termine all'interno di un lasso temporale prefissato, ripetendo le attività in maniera costante. In questo caso per ciascuno sprint è stato previsto un periodo di una settimana, all'interno della quale le varie fasi occupano un periodo pressoché costante. La struttura delle 4 iterazioni è riportata in Figura 4.6:

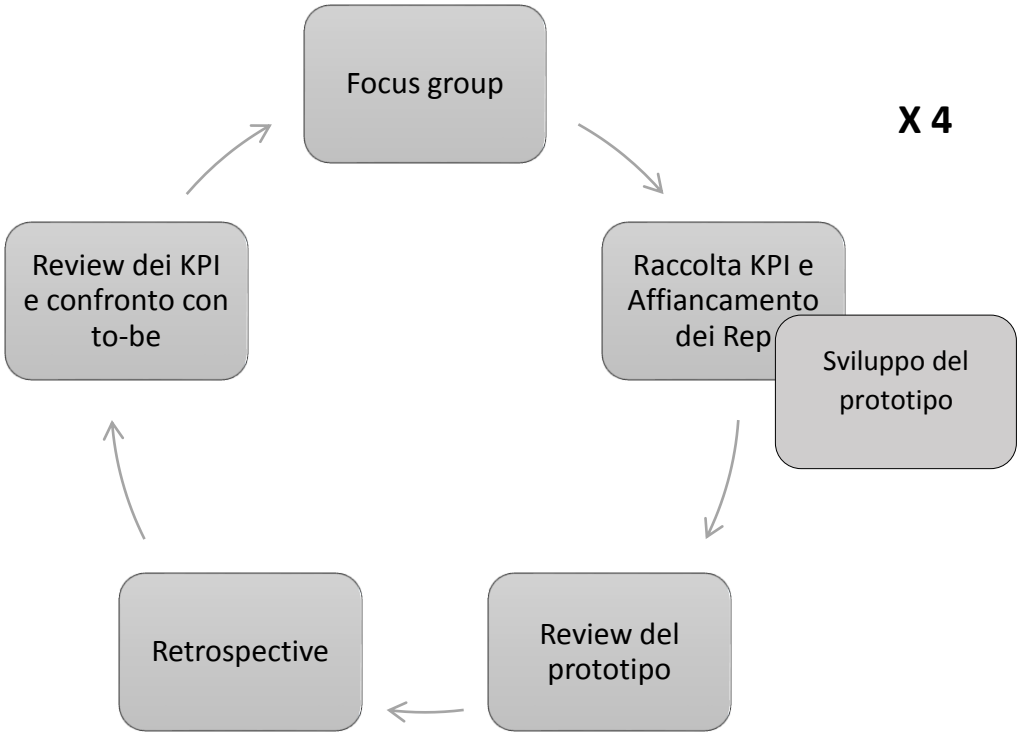

Figura 4.6: *Processo iterativo e incrementale*

Per ciascun processo iterativo è stata prevista la pianificazione temporale riportata in Figura 4.7:

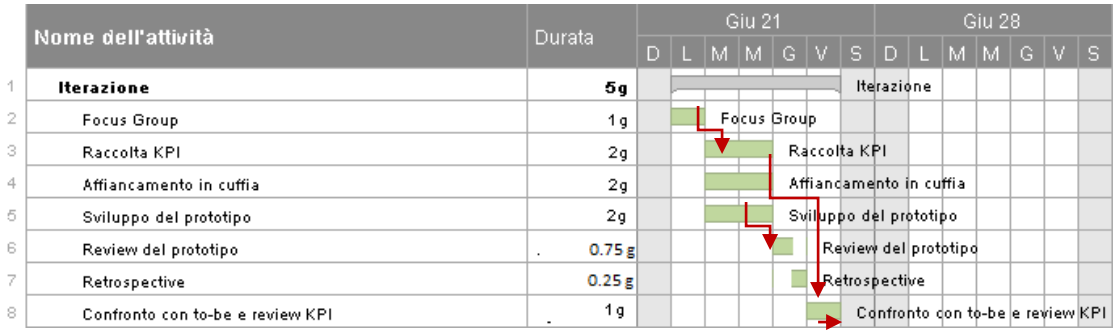

Figura 4.7: *Gantt di ciascuna iterazione*

Le frecce rosse rappresentano le relazioni presenti tra alcune fasi, ovvero la necessità di portare a termine un'attività per poter passare alla successiva. In particolare si ha:

1) Necessità di compilare il documento con le attività per ciascuno Use-Case durante il Focus Group, in modo tale da poter raccogliere i relativi KPI
2) Raccogliere tutti i KPI, in modo da far fare la review dei KPI da parte degli operatori stessi
3) Sviluppare il prototipo in tempo utile da poterne permettere una review per raccogliere i feedback degli end-user e dell'azienda Cliente
4) Avere la review dei KPI in tempi brevi in modo da poter fare il confronto col to-be su valori corretti

In conformità con l'approccio Agile, tali attività devono essere effettuate in tempi brevi, ed è estremamente importante che esse vengano concluse nelle tempistiche stabilite. Infatti all'interno di Sprint di una settimana, un ritardo di mezza giornata può provocare gravi problemi e il non raggiungimento degli obiettivi prestabiliti.

In seguito si riportano i dettagli sulle varie fasi del ciclo iterativo:

***Focus group**:* All'interno di ciascun Call Center la struttura organizzativa prevede la presenza di diversi team, composti da 10-12 persone, ciascuno rappresentato da un team leader. I team vengono a loro volta aggregati in gruppi, in base alla tipologia di problematiche delle quali si occupano.

Durante i Focus Group effettuati infatti è stata prevista la presenza, oltre che del Cliente, dei team leader e di alcuni membri dei team, in modo da rappresentare la gamma completa di tipologie di operatori e avere in input il maggior numero di requisiti possibile. In linea con la metodologia Agile infatti sono stati raccolti tutti i requisiti espressi durante l'incontro, in modo da individuarne poi solamente quelli ritenuti più importanti e focalizzarsi su essi.

L'incontro è stato a sua volta suddiviso in due sessioni:

1) *Presentazione della demo:* L'obiettivo di questa prima fase è quello di utilizzare la demo per raccogliere i requisiti che devono uscire liberamente dagli operatori partecipanti. Essi infatti sono liberi di esprimere qualsiasi commento e osservazione, di avanzare qualsiasi tipo di proposta per migliorare la soluzione che gli viene presentata e usare la loro esperienza di utilizzo dei sistemi presenti per identificare gli aspetti che dovrebbe presentare la soluzione to-be.

   L'articolazione di questa prima fase si svolge come un classico Focus Group, ovvero come un'intervista alla quale partecipa un gruppo di persone, su un ambito di indagine specifico. I partecipanti sono liberi di interagire fra loro, in modo tale da scambiarsi idee e confrontare esperienze diverse e diversi punti di vista. A differenza dell'intervista singola, in questo caso viene sfruttata la potenza del gruppo e la capacità di far emergere visioni e opinioni diverse, grazie alla possibilità di poter prendere spunto da quanto detto da altri. In questo modo viene generato un numero maggiore di idee e viene corso un rischio minore di perdere alcuni aspetti fondamentali.

Durante queste intervista, che si svolge nell'arco temporale di un'ora e mezza o due, partecipano circa 15 persone, un numero maggiore rispetto ai tipici Focus Group che prevedono il coinvolgimento di 6-10 persone. Questo apporta il beneficio di avere opinioni eterogenee e la raccolta di un numero maggiore di spunti, ma al tempo stesso conferisce un ruolo più importante al moderatore.
Tra il team di Accenture infatti è stata scelta una figura, esperta dei sistemi as-is, che fungesse da moderatore durante questi incontri. Il suo compito è quello di presentare il tema d'indagine, ovvero la demo realizzata fino a quel momento, e guidare e pilotare gli intervistati verso gli argomenti che più interessano.
Il moderatore infatti ha il compito di:

- Guidare la conversazione
- Favorire la discussione di tutti i partecipanti
- Evitare che la discussione sia dominata da un leader
- Mantenere una posizione di neutralità
- Evitare di esprimere le proprie opinioni e valutazioni

Egli inoltre deve assicurarsi che gli intervistati non divaghino, eludano o fraintendano il significato delle domande. Si tratta di un compito particolarmente difficile visto il numero dei partecipanti, infatti la scelta di questa figura è stata fatta ricadere su di una persona che grazie alle sue conoscenze sui sistemi as-is e sulla soluzione to-be, fosse in grado di capire e gestire le richieste e reindirizzare la conversazione ogni qualvolta i partecipanti stessero divagando.

Per quanto riguarda la demo presentata durante la prima iterazione non è stata altro che un prototipo estremamente basic, riportante i widget e le funzionalità definite in fase preliminare. Durante le iterazioni successive è stata presentato la versione incrementale del prototipo, ovvero una soluzione aggiornata di volta in volta con le richieste provenienti dalla fase precedente.

Durante questa prima riunione, della durata di circa 2 ore, vengono presi appunti dal moderatore e da altre 3 persone del Team, generalmente il Manager che ricopre il ruolo di Scrum Master e un membro per ciascuno dei due sottogruppi, business e tecnico, del Team Accenture, che ricoprono quindi il ruolo di "Note Taker". Finite le due fasi di questo incontro, queste persone si ritrovano per aggregare insieme le fedeli sbobinature prese durante questa intervista e cominciare una loro analisi. Quanto raccolto viene infatti "filtrato" dalle richieste impossibili da sviluppare sull'applicativo, grazie alle conoscenze tecniche e relative alle policy aziendali dell'azienda Cliente di alcuni membri del Team e successivamente vengono scomposti e segmentati i requisiti rimanenti, in modo da ricondurli a macro-aree di appartenenza. Tali aree sono riconducibili ai vari widget presenti sull'applicativo, in modo tale che le richieste seguano un chiaro criterio di ordinamento.

Questi requisiti vengono quindi riportati all'interno di un apposito documento, il quale servirà da sostegno agli sviluppatori del prototipo e sul quale dovrà essere tempestivamente aggiornato lo stato di fattibilità dei requisiti stessi.

Viene riportato in Figura 4.8 un esempio di tale documento, sul quale sono evidenti, oltre i requisiti riportati per ciascun widget, dei semafori per indicare lo stato in cui si trovano tali requisiti:

- Implementato sul prototipo
- In fase di sviluppo
- Necessaria analisi di fattibilità tecnica

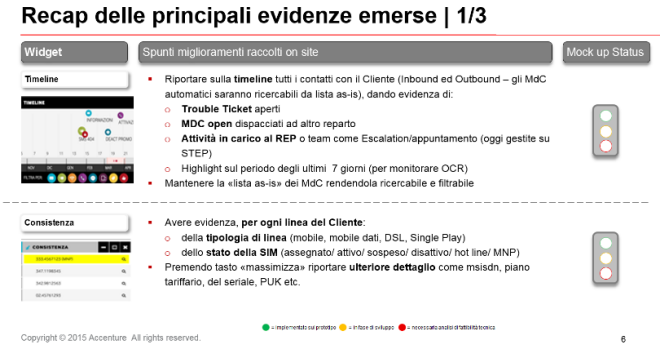

Figura 4.8: *Documento di raccolta requisiti*

2) *Compilazione delle attività degli Use Case:* questa seconda fase viene dedicata alla compilazione di una parte del file Excel visto in precedenza. In particolare, grazie all'aiuto dei Rep, devono essere individuate le attività di cui è composto ciascuno Use Case, quali dati devono essere presi in input, su quali sistemi vengono svolte attualmente le varie azioni e se sono presenti dei pain point evidenti. Come già detto, questa attività, della durata di 3/4 ore, serve sia per mappare la situazione as-is in modo da ribaltare il flow operativo sulla soluzione to-be, che per stimolare ulteriori requisiti da parte degli operatori.

   Questa seconda parte dell'intervista è maggiormente strutturata in quanto deve portare alla compilazione di tutti i campi, per ciascuno Use-Case, contenuti nel documento preparato. In questo caso infatti gli operatori devono usare la loro esperienza in modo più oggettivo e razionale rispetto alla fase precedente. Mentre nella prima parte dell'incontro sono richieste anche le sensazioni degli end-user per capire in che modo reagiscono istintivamente all'applicativo, in questa seconda sezione è richiesto un approccio più rigoroso per andare a capire quali passi vengono seguiti durante la gestione delle diverse tipologie di chiamata.

   Anche in questo caso risulta fondamentale la figura del moderatore, il quale deve prevenire che la discussione divaghi eccessivamente e deve contribuire a fare ordine nella compilazione della sequenza delle attività, qualora ci siano conflitti tra i partecipanti.

   Oltre al moderatore, durante questa fase deve essere prevista la presenza di una persona del Team che compili il documento, che deve essere proiettato in modo tale che sia ben visibile da tutti i partecipanti, al fine di evitare errori durante la trascrizione delle fasi. La scelta di non far compilare il documento al moderatore, serve a consentirgli di concentrarsi maggiormente sulla conversazione, in modo da ottenere i risultati desiderati in tempi ragionevoli.

La problematica maggiore riscontrata infatti durante questa seconda fase, è stata quella di non riuscire a completare in tempi ragionevoli il documento, a causa dell'elevato numero di partecipati che esprimono la loro diversa opinione sulla sequenza delle attività da svolgere per gestire le telefonate.

Questo ha portato spesso a dover allungare le tempistiche di questa fase, al fine di raggiungere l'obiettivo preposto e in modo tale da poter passare alla fase successiva del processo iterativo.

Raccolta KPI e affiancamento dei Rep: Durante questa fase, svolta in parallelo allo sviluppo del prototipo, vengono raccolti i dati relativi ai KPI individuati (numero schermate, numero di click e AHT) per ciascuna attività di ciascuno Use Case. Per reperirli devono essere seguite queste attività step-by-step sui dispositivi degli operatori, in modo da confrontarli successivamente con i valori lato to-be.

Oltre al reperimento dei valori per i KPI, viene fatto l'affiancamento in cuffia per vedere realmente cosa accade durante la gestione delle chiamate. In questo modo possono essere recepite anche le azioni che vengono fatte quotidianamente dagli operatori e che possono differire da quelle pensate dalle procedure per loro predisposte. Grazie a questo possono essere pensate delle soluzioni migliorative per il nuovo applicativo, che seguano di più il lavoro reale del Rep.

Durante questa attività infatti viene osservato l'operatore durante il suo lavoro abituale, senza prenderne veramente parte. I membri del Team che seguono questa attività infatti si collegano con le cuffie al pc dell'end-user e ascoltano in silenzio la telefonata, prendendo appunti sull'utilizzo dei sistemi durante la gestione della problematica e sul modo di interazione tra operatore e Cliente. Questo tipo di attività permette di sviluppare una visione "dal di dentro" del lavoro svolto nei Call Center e di accrescere la comprensione sulle attività svolte.

Il periodo di tempo per svolgere questa attività varia tra le 2 e le 3 ore, per permettere al membro del Team di inserirsi nell'ambiente naturale di lavoro degli operatori,

prendere familiarità con il loro lavoro, in tutti i suoi aspetti, descrivere le azioni che svolgono e comprenderne le motivazioni.

Questo lavoro deve consentire al Team di familiarizzare con la situazione as-is, per andare a contribuire in modo più efficace nello sviluppo dell'applicativo.

Sviluppo del prototipo: Questa attività fatta in parallelo a quella descritta sopra, deve essere svolta, conformemente all'Agile, in tempi fissi molto rapidi. Questo lavoro deve essere concluso in un paio di giorni, in modo da presentare il prototipo al Cliente e agli end-user per ricevere dei feedback a riguardo durante la visita on-site. Nel caso non riuscissero ad essere sviluppati tutti i requisiti in tempo per la raccolta dei feedback, il team dovrebbe però garantire lo sviluppo del prototipo prima dell'inizio dell'iterazione successiva. Il prototipo infatti viene sviluppato in modo incrementale, partendo dalla soluzione finita nell'iterazione precedente e andando ad implementare i requisiti recepiti durante il focus group.

Durante lo sviluppo devono essere effettuate delle analisi di fattibilità per verificare che alcune richieste possano essere effettivamente assolte. Il tracciamento di quello che è presente sul prototipo, quello che è in fase di sviluppo e quello che abbisogna di un'analisi di fattibilità viene raccolto in un apposito documento e aggiornato costantemente.

Una volta verificati i requisiti che possono essere effettivamente implementati, le fasi di cui si compone lo sviluppo del prototipo sono le seguenti:

- *Developing:* Si tratta dello sviluppo di porzioni di codice che andranno a comporre l'applicativo
- *Unit test:* Durante questa fase viene fatto il test delle porzioni di codice sviluppate, al fine di verificare il corretto funzionamento delle singole unità
- *Deployment:* Le porzioni di codice sviluppate e funzionanti vengono quindi implementate all'interno di un "*test environment",* installato su un server e configurato su una piattaforma web

- *System test:* Infine si procede a fare una verifica delle funzionalità del sistema, in modo da testare l'adeguatezza del software nella sua interezza

Review del prototipo: Al termine della visita on-site viene tenuto un meeting di un paio d'ore a cui partecipano gli end-user, che hanno partecipato al focus group, e i rappresentanti HQ business e technology. Il team presenta il nuovo prototipo sul quale sono state inserite le richieste fatte qualche giorno prima, in modo da raccogliere gli eventuali feedback, sia degli utilizzatori che del cliente stesso. In questo modo vengono individuati subito eventuali misunderstanding avvenuti durante il focus group. Inoltre, l'end-user vedendo concretamente quanto aveva richiesto riesce a rendersi conto fin da subito di eventuali mancanze o ulteriori miglioramenti apportabili alla nuova soluzione.

Come si evince da questa fase e dal focus group, in linea con i principi dell'Agile, il prototipo ha una funzione principale. Esso infatti è il protagonista durante questo percorso iterativo e serve da interfaccia tra il team e il cliente/end-user, in modo da permetterne una collaborazione efficace. Durante tutte le fasi viene quindi data un'importanza maggiore al prototipo piuttosto che alla documentazione, che passa in secondo piano come elemento di supporto.

Retrospective: Finita la review del prototipo, viene rivisto, da parte del solo team, il percorso svolto durante la visita, in modo da ricercare soluzioni migliorative da mettere in atto per la visita successiva. Tali soluzioni possono riguardare qualsiasi attività, dalla gestione del focus group alla compilazione della documentazione.

Review KPI e confronto con il to-be: Come ultimo step del ciclo si ha la review dei KPI che viene svolta grazie alla collaborazione di alcuni team-leader dei Call Center. Viene infatti inviato il file con i dati raccolti on-site, in modo che i valori raccolti possano essere verificati direttamente dagli operatori, i quali possono ripercorrere più volte le

attività in modo da assicurarsi della loro correttezza ed eventualmente modificare i dati col valore esatto.

Fintanto che tale lavoro non viene svolto si ha l'impossibilità di passare alla fase successiva, in quanto verrebbero confrontati dati scorretti con i valori to-be.

Una volta avuto il check dei parametri raccolti, vengono selezionati tra i vari Use Case, quelli di maggiore rilevanza e vengono confrontate le azioni svolte sugli applicativi as-is, osservate durante la visita, con le azioni che verrebbero svolte sulla nuova soluzione grazie all'utilizzo della versione to-be del prototipo.

Per farlo vengono raccolti tutti i valori all'interno di un apposito documento, riportato in Figura 4.9:

						KPIs As-Is			
Frequenza	Use Case	Customer need	Attività del rep	Info / dati necessari	Sistemi sorgente dati	# clicks	# schermate	ANT (Average Navigation Tim	AHT (minuti)
ALTA	Verifica campagne	Cliente ha ricevuto messaggio / chiamata / direct mail per promozione offerta e chiama per avere chiarimenti (principalmente su temi di contabilità di tempi e con promozioni già attive) e/o chiederne attivazione	Se cliente fa riferimento a chiamata ricevuta fa check in ATLAS per verificare esistenza contatto outbound		ATLAS	11	2	20	8
			Se cliente fa riferimento a messaggio / direct mail ricevuta fa check in CCM		CCM (tab dedicato "Inviti" TDC, RTM)	9	2	10 sec per TDC - 5 sec per Inviti	
			Verifica stato attivazione promo / offerta accettata		BTC (Order Manager mobile)	12	2	39	
			Attivazione promo da parte del rep (se non già in attivazione / attiva)		BESFO (bundle), OCA (mobile pure)	10	2	OCA 65 BESFO 33	
			Ricerca su chiamate effettivamente ricevute dal cliente (e controllare il dealer / partner che ha contattato il cliente) - se non trovo info su ATLAS / CCM (utilizzato esclusivamente dai Team Leader). Anche per info campagne sui social media		EAGLE (accedibile via JACK e solo dai Team	7	2	non utilizzato da rep	
			Inserimento Motivo di contatto (MdC)			10	2	15	
						59	12		

		KPIs To-be												
Frequenza	Use Case	ANT actual	# clicks to be	ANT to be (scen 0)	Delta ANT (scen 0)	% saving ANT (scen 0)	ANT to be (scen 1)	Delta ANT (scen 1)	% saving ANT (scen 1)	AHT actual	AHT to be (scen 0)	AHT to be (scen 1)	% Saving (scen 0)	% Saving (scen 1)	Motivo efficientamento
ALTA	Verifica campagne	20	1	3	-17	-85%	1	-19	-95%						filtro in timeline
		10	0	0	-10	-100%	0	-10	-100%						informazione presa col punto precedente
		39	1	3	-36	-92%	1	-38	-97%						widget in homepage
		65		65	0	0%	65	0	0%						sistema as-is
		15	3	5	-10	-67%	3	-12	-80%						widget in homepage
		149		76	-73	-49%	70	-79	-53%	480	407	480	-15%	0%	

Figura 4.9: *Documento di confronto KPI as-is vs to-be*

Per il calcolo dell'AHT to-be, viene introdotto un nuovo KPI: l'Average Navigation Time (tempo di navigazione media da parte del Rep per ciascuna attività) calcolato grazie a dei software con strumenti di reportistica integrati. Sono stati successivamente ipotizzati due scenari to-be possibili: best-case scenario e worst-case scenario. La percentuale di miglioramento dell'ANT viene calcolata come:

$$\%Miglioramento\ ANT = \frac{ANT_{to-be}}{ANT_{as-is}} - 1$$

Tramite l'ANT vengono poi individuate le percentuali di miglioramento dell'Average Handling Time per ciascuno scenario nel modo sotto riportato:

$$AHT_{to-be} = AHT_{as-is} - (ANT_{as-is} - ANT_{to-be})$$

$$\%Miglioramento\ AHT = \frac{AHT_{to-be}}{AHT_{as-is}} - 1$$

4.2.3.1 Ruoli durante le fasi iterative

Durante le fasi del progetto sopra descritte sono stati definiti dei ruoli ben precisi, alcuni dei quali sono rintracciabili nei ruoli visti nello Scrum.

In primo luogo all'interno dell'HQ del Cliente è stata individuata una persona che fungesse da Product Owner del processo e che si occupasse di fare da tramite tra il team di sviluppo dell'applicativo e l'area business della società di telecomunicazioni. Il suo compito principale è quello di riunire in un documento unico i requisiti provenienti dal lato business e mantenerlo aggiornato e coerente con quanto uscito durante le visite on-site. Tale figura infatti partecipa al focus group all'inizio di ogni interazione e alla review finale del prototipo. Dopo avere fornito dei requisiti di base all'inizio del progetto, facendosi portavoce dell'area business, aggiorna il documento grazie a quanto espresso dagli end-user nelle riunioni di cui sopra e a quanto mostrato dagli aggiornamenti del prototipo. Egli è il responsabile ultimo della lista dei requisiti finali e dell'ordine di priorità con cui dovranno essere sviluppati nella versione finita del

software. Durante le riunioni inoltre si occupa di regolare il rapporto tra team di sviluppo e operatori dei Call Center in modo da agevolarne la collaborazione e facilitarne la comunicazione.

La seconda figura definita è il Manager di Accenture, il quale è assimilabile allo Scrum Master. Egli si presenta come la guida del processo di raccolta dei requisiti e sviluppo del prototipo e partecipa durante tutte la visite on-site per gestire le fasi delle 4 iterazioni. Egli infatti si pone come portavoce del team di sviluppo durante l'interazione con il product owner recependone le volontà e illustrandogli lo svolgimento del lavoro svolto presso i Call Center. Inoltre si assicura che il team esegua le attività individuate all'interno di ogni incremento, senza incontrare ostacoli e nei tempi prestabiliti. Tutto questo deve avvenire senza imporre la sua autorità, ma lasciando libertà al team durante lo svolgimento dei vari compiti, in modo da creare un team responsabilizzato e autonomo.

Infine si ha il team di persone che si recano presso i Call Center per reperire requisiti, KPI e sviluppare il prototipo. Si tratta di un team che rispecchia le 2 qualità principali dell'Agile: autonomo e cross-funzionale. Le persone al suo interno infatti sono responsabilizzate in modo tale da svolgere i compiti in modo autonomo, senza il bisogno di un'autorità che stressi i tempi e controlli costantemente gli output del loro lavoro. Inoltre all'interno del gruppo sono presenti tutte le competenze necessarie per portare avanti il lavoro, senza bisogno di aiuti esterni. Il team è composto di 7 persone con ruoli diversi e diviso essenzialmente in sottogruppi. Il primo è formato da 3 persone le quali si occupano di reperire i requisiti durante il focus group, elaborare il documento da aggiornare durante lo sviluppo del prototipo e raccogliere i KPI. Il secondo sottogruppo è formato da 3 esperti dell'applicativo che si occupano di trasformare i requisiti raccolti in requisiti tecnici, farne eventuali analisi di fattibilità e sviluppare il codice per costruire il prototipo. Infine esiste una figura esperta dei sistemi as-is, che funge soprattutto da supporto ai due sotto team sopra individuati, funge da moderatore durante gli incontri col Cliente e gli end-user grazie alle sue conoscenze e riesce ad arricchire le sue conoscenze grazie agli affiancamenti in cuffia.

4.3 Risultati ottenuti durante le visite on-site

Il processo di raccolta dei requisiti presso i Call Center è iniziato utilizzando una versione basilare dell'applicativo, riportante alcune funzionalità individuate in fase preliminare.

Tale soluzione, visibile in Figura 4.10, riporta su un'unica landing page i seguenti elementi, che si popolano con le informazioni del cliente chiamante:

- *Search box*: Barra di ricerca su cui compare il numero del cliente chiamante. Serve inoltre a contestualizzare l'intera pagina in base al tipo di cliente e al tipo di chiamata ricevuta
- *Timeline*: Widget che mostra i contatti inbound e outbound avvenuti nell'ultimo periodo tramite chiamata, mail, SMS, chat, app e web
- *Contatore linea*: Widget che presenta 3 contatori per voce, traffico e dati e mostra il credito residuo totale del cliente e lo storico ricariche
- *Invio*: Widget che consente all'operatore di inviare comunicazioni al cliente
- *Promo Attive*: Widget sul quale sono presenti il piano tariffario e le offerte attive sul device del cliente con relativa data di attivazione e scadenza
- *Consistenza*: Widget che presenta tutte le SIM e linee del Cliente
- *Servizi*: Widget riportante altri servizi attivi sulla SIM del cliente
- *NBA*: Widget che suggerisce all'operatore le offerte da proporre al Cliente

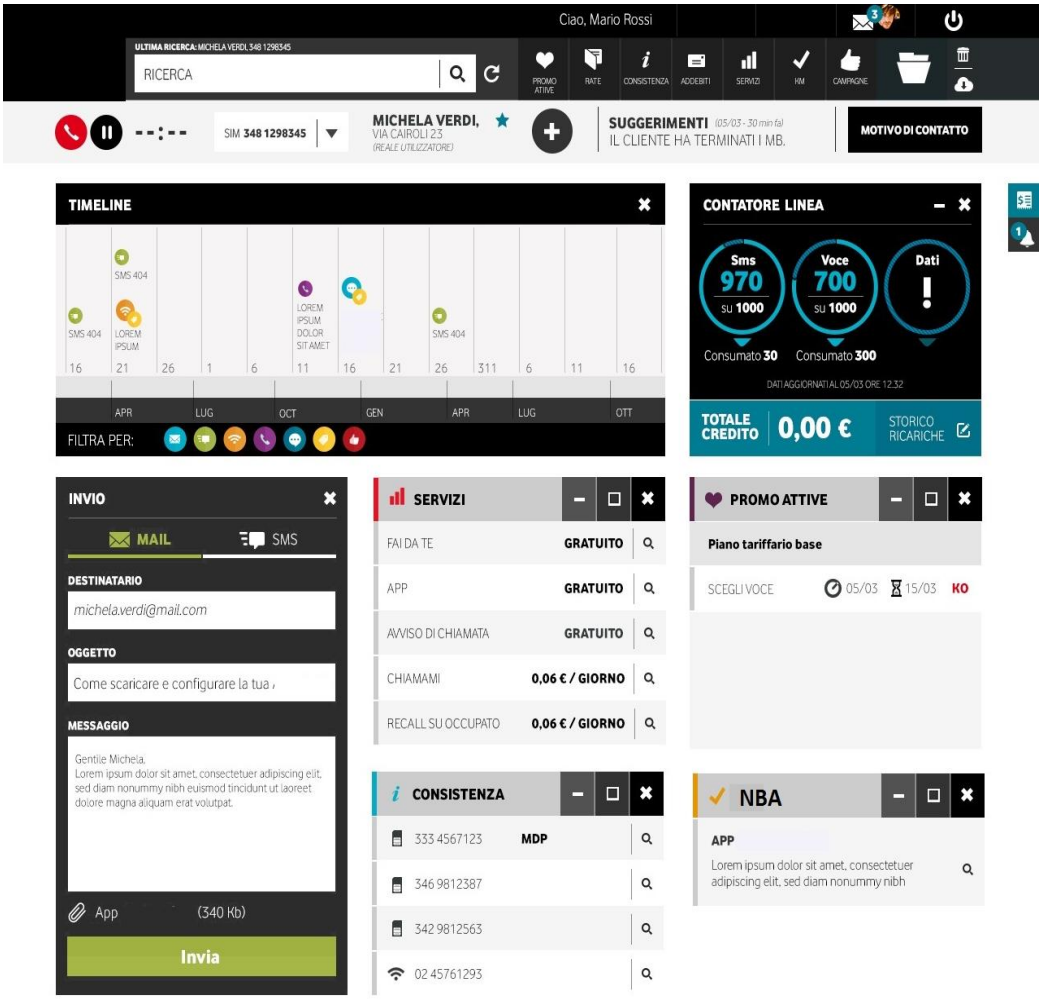

Figura 4.10: *Prototipo inziale*

Pisa:

Durante la visita presso il Call Center di Pisa è stato svolto, come da processo iterativo, il Focus Group durante il quale sono emersi ulteriori requisiti da sviluppare sull'applicativo. Tali requisiti sono riportati nel documento mostrato in Figura 4.11 e hanno riguardato specialmente richieste relative all'ambito Consumer Mobile Prepaid:

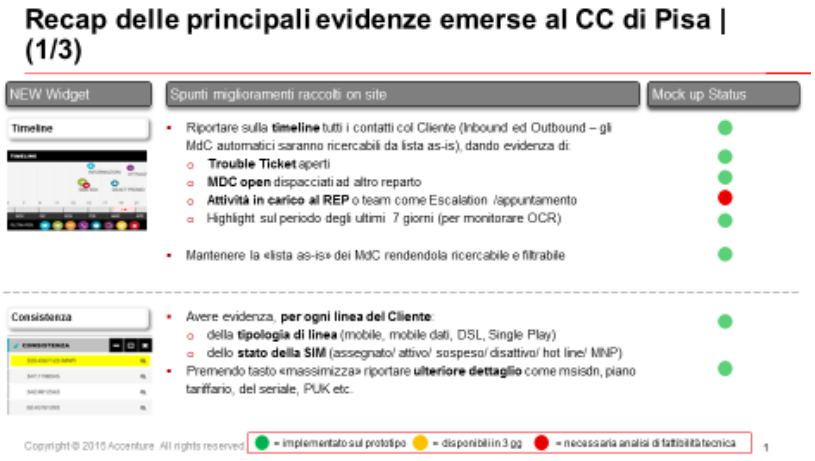

Figura 4.11: *Requisiti raccolti a Pisa*

Come è visibile in figura, oltre a riportare i requisiti raccolti, il documento presenta degli indicatori utilizzati per mostrare lo stato di implementazione delle richieste sull'applicativo. Tali "semafori" sono stati aggiornati real-time durante la visita, per tenere traccia dello stato di avanzamento dello sviluppo del prototipo. Il colore verde segnala una funzionalità già implementata sul prototipo, il giallo indica una funzionalità in fase di sviluppo e il rosso viene assegnato a quei requisiti che richiedono un'analisi di fattibilità tecnica.

Grazie a ai requisiti raccolti, sono state aggiunte ulteriori funzionalità ai vari widget presenti sull'applicativo e in particolare è stato creato ex-novo il widget riportato in Figura 4.12 e sono state aggiunte informazioni sul widget in Figura 4.13:

- *Widget Movimentazione*: riportante gli ultimi addebiti sulla SIM del Cliente dovuti a messaggi, telefonate, internet ecc. ed eventuali accrediti dovuti ad esempio a ricariche telefoniche.

  L'esigenza di avere questo widget è nata dalla necessità di avere uno strumento sul quale fosse possibile visualizzare velocemente tutti gli accrediti e gli addebiti effettuati sulla SIM del Cliente, in modo da poter gestire Use Case come "Non mi torna il credito". In questo modo la problematica può essere risolta in tempi minori, avendo tutte le informazioni immediatamente accessibili.

Figura 4.12: *Widget movimentazioni*

- Visualizzazione sul Widget Contatori di informazioni relative ai servizi roaming con un focus sulle attività di messaggistica, chiamate e traffico dati.

  Questa richiesta è scaturita a causa del frequente numero di problematiche riguardanti il credito sulla SIM, dovute al traffico voce, sms e dati sotto condizioni di roaming. In questo modo l'end-user ha la possibilità di visualizzare a colpo d'occhio se e quanto traffico è stato generato in roaming.

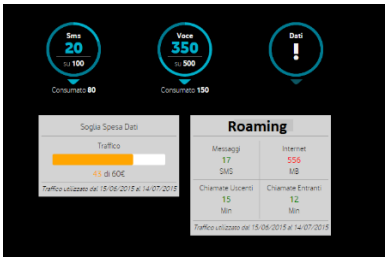

Figura 4.13: *Informazioni su Servizi Roaming*

Durante la visita a Pisa, oltre al recepimento dei requisiti e al relativo sviluppo del prototipo, sono stati raccolti i kPI relativi agli UC individuati per questo CC. Se ne riporta in Tabella 4.4 un esempio:

UC esemplificativo	Attività	Numero di click	Numero di schermate	AHT as-is (secondi)	ANT as-is (secondi)
Non mi torna il credito	Verifica credito residuo	13	3	420	136
	Verifica dettaglio chiamate / traffico generato	11	3		
	Verifica servizi a pagamento	8	2		
	Verifica esistenza promo attive non richieste dal client	3	2		
	Inserimento Motivo di contatto (MdC)	10	2		

Tabella 4.4: KPI per “Non mi torna il credito” as-is

Successivamente è stato svolto il confronto tra i vari indicatori riportato in Figura 4.11, utilizzando la formula descritta precedentemente per il calcolo del AHT to-be:

	ANT	% Savings	AHT	% Savings	#Click
As-Is	136		420		45
To-be (best)	62	-53%	346	-18%	5
To-be(worst)	78	-49%	362	-14%	5

Tabella 4.5: KPI per “Non mi torna il credito” as-is vs to-be

<u>Bologna</u>:

Anche durante la visita on-site presso il CC di Bologna è stato compilato il documento, riportato in Figura 4.14, contenente tutti i requisiti espressi dagli end-user. Anche in questo caso le richieste sono state relative al mondo Consumer Mobile Prepaid, ma con un focus particolare su chiamate relative a problemi tecnici e a Clienti nuovi.

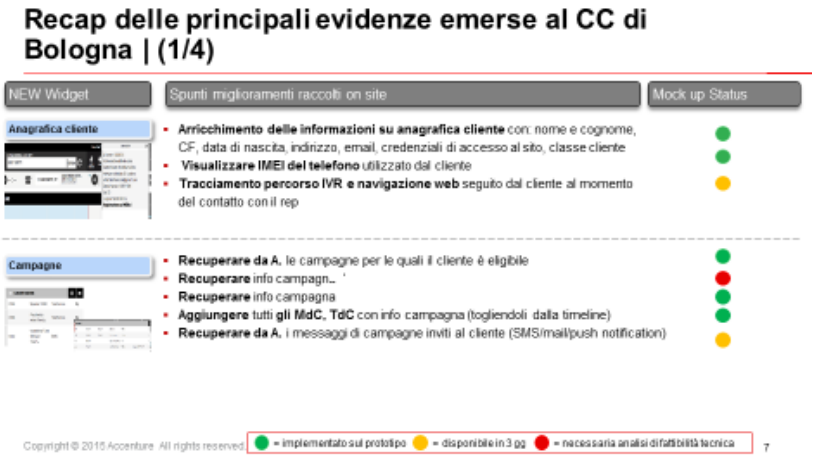

Figura 4.14: *Requisiti raccolti a Bologna*

Tra le varie richieste raccolte è nata l'esigenza di aggiungere due nuovi widget, visibili in Figura 4.15 e in Figura 4.16:

-*Widget campagne*: riportante tutte le offerte proposte al Cliente attraverso svariati canali di comunicazione: messaggi, chiamate ecc. Questa esigenza è nata dalla necessità di avere uno strumento per verificare rapidamente se al Cliente sono state effettivamente proposte le offerte per cui ha chiamato il Call Center senza dover aprire più sistemi diversi per andare a rintracciarle tutte, rallentando notevolmente i tempi di gestione della chiamata.

Figura 4.15: *Widget Campagne*

-*Widget semafori:* il quale permette di indicare in modo evidente all'operatore alcune particolari problematiche. Questa richiesta è nata poiché durante i Focus Goup è sorta l'esigenza da parte degli end-user di avere una segnalazione evidente, appena contestualizzata la pagina su uno specifico Cliente, la presenza di particolari problematiche. Tra queste situazioni, le più tipiche sono: la manifestata volontà da parte del Cliente, in occasioni precedenti, di voler cambiare Operatore, la presenza di una SIM scaduta o l'evidenza di offerte pensate ad hoc per il Cliente.

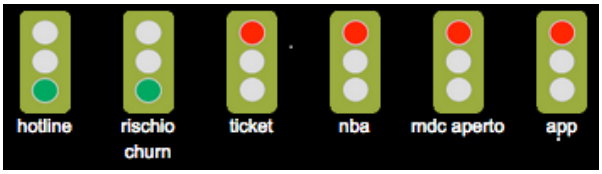

Figura 4.16: *Widget Semafori*

Oltre a questi requisiti, è emersa la necessità di avere informazioni riguardo al cliente chiamante, in una seziona dedicata chiamata *Anagrafica.* Questa richiesta è emersa poiché spesso durante le chiamate si ha necessità di avere sotto mano i dati del Cliente sia per questioni di riconoscimento, che di invio di documenti o per accertarsi dell'accettazione di alcuni particolari condizioni. (Figura 4.17)

Inoltre è sorta l'esigenza di mostrare con colori diversi l'avvicinamento al limite della soglia nel *Widget Contatori* (Figura 4.18). Questa richiesta è stata dovuta al fatto che spesso l'operatore, stando molte ore davanti al pc, ha bisogno di alcuni "escamotage" per aiutare a focalizzare lo sguardo su alcune informazioni, che altrimenti rischierebbero di passare inosservate. Per questo la scelta di evidenziare il consumo effettivo, rispetto al totale, con colori diversi risulta essere un valido aiuto per visualizzare l'informazione.

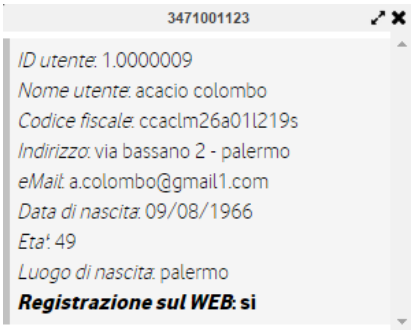

Figura 4.17: *Anagrafica Cliente*

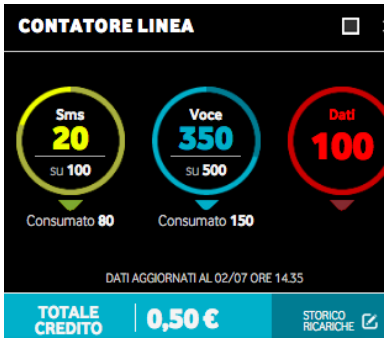

Figura 4.18: *Avvicinamento soglie*

Infine al termine della visita, in seguito ai feedback raccolti mostrando il prototipo si è deciso di modificare il *Widget Semafori* in *Widget Alert* in modo da consentirne un uso più pratico. Per lo scopo per il quale il widget è stato pensato, risulta più conveniente avere una lampadina che cambia stato in on/off, come è visibile in Figura 4.19, piuttosto che un semaforo con 3 diversi stati di avanzamento.

Figura 4.19: *Widget Alert*

Anche durante la visita presso il CC di Bologna sono stati raccolti i KPI per le varie casistiche, di cui si riporta un esempio in Tabella 4.6:

UC esemplificativo	Attività	Numero di click	Numero di schermate	AHT as-is (secondi)	ANT as-is (secondi)
Registrazione al sito	Invito alla registrazione sul sito / app (no uso di sistemi)			360	122
	Verifica se il cliente è registrato, quante SIM sono registrate con lo stesso user	5	4		
	In caso di blocco (dopo 3 tentativi) --> sblocco	1	1		
	Motivo di contatto	10	2		

Tabella 4.6 KPI per “Registrazione al sito” as-is

Durante il confronto con il to-be sono emersi i seguenti risultati, esposti in Figura 4.7:

	ANT	% Savings	AHT	% Savings	#Click
As-Is	122		360		16
To-be (best)	67	-45%	305	-15%	5
To-be(worst)	75	-39%	313	-13%	5

Tabella 4.7: KPI per "Registrazione al sito" as-is vs to-be

Catania:

Nel corso della visita al Call Center di Catania sono state recepite le osservazioni degli operatori che si occupano della clientela Mobile Postpaid e in particolar modo al mondo delle chiamate per problematiche di tipo tecnico e riguardanti gli abbonamenti. Anche in questo caso le richieste sono state raccolte nell'apposito documento mostrato in Figura 4.20:

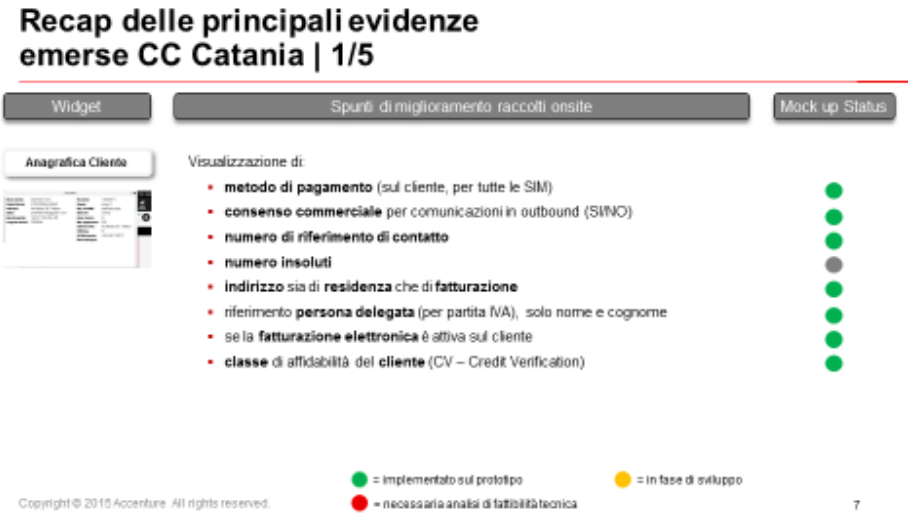

Figura 4.20: *Requisiti raccolti a Catania*

Durante questo sprint sono stati sviluppati due nuovi widget, riportati in Figura 4.21 e in Figura 4.22:

-*Widget rate:* riportante informazioni relative alle rate del device acquistato tramite abbonamento dal Cliente. L'esigenza di avere questo widget è nata poiché nel Call Center di Catania vengono gestite le problematiche di Clienti che prendono i telefoni in abbonamento, quindi risulta fondamentale per l'operatore avere evidenza, in modo facilmente accessibile, delle informazioni riguardanti le rate del telefono che hanno acquistato. Questo permette di gestire in modo facile e veloce tutte le telefonate relative alla richiesta di informazioni sul numero di rate, il totale ancora da pagare, la scadenza del vincolo ecc.

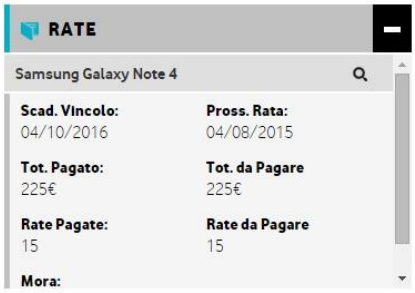

Figura 4.21: *Widget Rate*

- *Widget fatture:* il quale mostra le fatture del cliente relative all'ultimo periodo, evidenziando in rosso quelle non pagate. Anche in questo caso l'esigenza è nata dal fatto che molte telefonate dei Clienti in abbonamento, sono relative a problematiche con la fattura. Queste possono variare dalla non ricezione della fattura alla non comprensione di alcune spese su di essa riportate. Per questo motivo risulta fondamentale per l'operatore avere evidenza dei documenti in pdf, semplicemente cliccano sulla relativa fattura, in modo da poter spiegare al Cliente il perché di alcuni addebiti.

Inoltre l'operatore ha la possibilità di vedere a colpo d'occhio l'eventuale presenza di fatture non pagate, semplicemente notando la presenza del colore rosso sul widget, in modo da poter informare prontamente il Cliente.

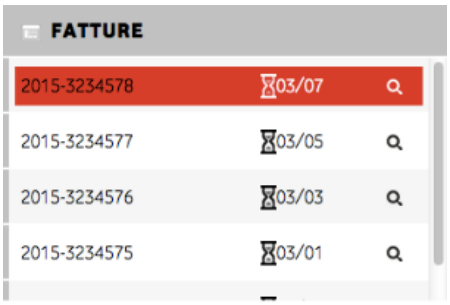

Figura 4.22: *Widget Fatture*

Inoltre è stato incrementato il requisito espresso a Bologna, di avere un *Anagrafica* Cliente, aggiungendovi ulteriori informazioni che potessero risultare utili durante le chiamate dei Clienti in abbonamento e non solo di tipo pre-paid, fino ad ottenere quanto mostrato in Figura 4.23:

Figura 4.23: *Anagrafica Cliente*

Per quanto riguarda la raccolta dei KPI, si riporta un caso esemplificativo, di un tipo Use-Case di un CC che si occupa di abbonamenti:

UC esemplificativo	Attività	Numero di click	Numero di schermate	AHT as-is (secondi)	ANT as-is (secondi)
Aggiornamento servizi in abbonamento	Apertura fattura	3	2	390	127
	Eventuale disattivazione del servizio non richiesto	5	3		
	Verifica della presenza di offerte suggerite dal sistema	1	2		
	Trasferimento della problematica a un altro CC	3	1		
	Inserimento Motivo di contatto (MdC)	10	2		

Tabella 4.8: KPI per "Aggiornamento servizi in abbonamento" as-is

Nel confronto col to-be si ha quanto riportato in Tabella 4.9:

	ANT	% Savings	AHT	% Savings	#Click
As-Is	127		390		22
To-be (best)	73	-42%	336	-14%	6
To-be(worst)	82	-35%	345	-12%	6

Tabella 4.9: KPI per "Aggiornamento servizi in abbonamento" as-is vs to-be

Ivrea:

Al Call Center di Ivrea è stato condotto il Focus Group per recepire le richieste relative al mondo della rete fissa. Anche in questo caso è stato redatto il documento dei requisiti, riportato in Figura 4.24:

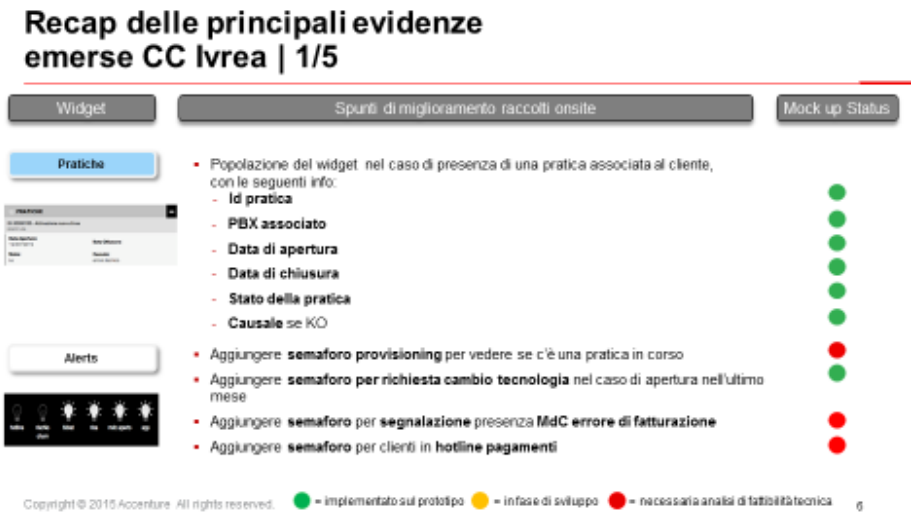

Figura 4.24: *Requisiti raccolti a Ivrea*

Durante questa iterazione è stato sviluppato il seguente widget (Figura 4.25):

- *Widget Pratiche:* riportante le informazioni relative alle pratiche di uno specifico Cliente. L'esigenza di questo widget è emersa durante questo Focus Group in quanto esso è relativo alle pratiche associate all'attivazione di una nuova linea fissa e per tanto solamente durante la visita presso il CC che si occupa del mondo del fisso, poteva emergere tale esigenza. Grazie a questo widget l'end-user riesce a verificare velocemente se esistono delle pratiche relative alla linea di un Cliente e qual è il loro stato.

Figura 4.25: *Widget Pratiche*

Sono stati inoltre aggiunti dei contatori e dei semafori riferiti al mondo della rate fissa e il widget consistenze è stato differenziato a seconda che l'asset sia una SIM mobile o una linea fissa (Figure 26,27 e 28), in modo da tener presenti anche le attività svolte per la gestione di problematiche esclusivamente legate a questo mondo.

Figura 4.26: *Widget Contatori Rete Fissa*

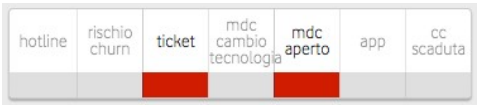

Figura 4.27: *Widget Alert*

Figura 4.28: *Widget Consistenza: Rete Fissa e Rete Mobile*

La differenza nel layout di questi ultimi widget è dovuta al fatto che durante questo sprint, ma basandosi anche su richieste emerse durante le visite precedenti, è stata modificata la landing page, sviluppando una versione dell'applicativo con un nuovo look&feel, visibile in Figura 4.29:

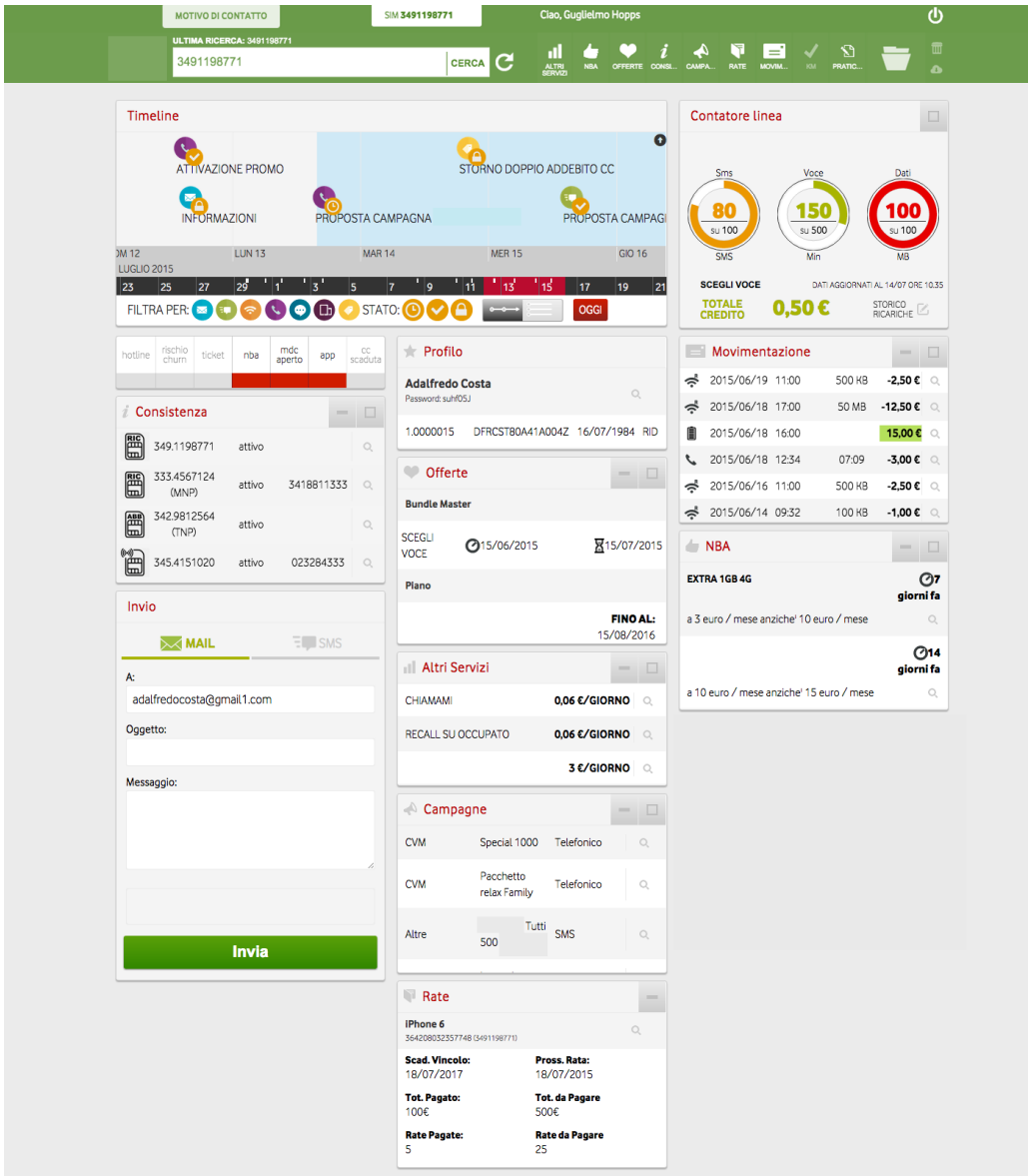

Figura 4.29: *Nuovo look&feel*

La modifica del layout della pagina è dovuta al fatto di aver recepito l'esigenza, mostrata dagli end-user, di dover lavorare molte ore al giorno davanti al pc e di prediligere colori più tenui che stancassero meno la vista.

Infine anche durante questa visita si sono mappati i processi as-is, raccogliendo i relativi KPI, riportati in modo esemplificato in Tabella 4.10:

UC esemplificativo	Attività	Numero di click	Numero di schermate	AHT as-is (secondi)	ANT as-is (secondi)
Fattura ADSL/Fibra	Verifica del piano telefonico	1	1	330	174
	Apertura fattura per check col Cliente	3	3		
	Se c'è presenza di errore si procede al riaccredito	3	3		
	Inserimento Motivo di contatto (MdC)	10	2		

Tabella 4.10: KPI per "Fattura ADSL/Fibra" as-is

Confrontandolo con l'ipotesi di to-be si ha quanto riportato in Tabella 4.11:

	ANT	% Savings	AHT	% Savings	#Click
As-Is	174		390		17
To-be (best)	51	-71%	267	-31%	3
To-be(worst)	87	-50%	303	-22%	3

Tabella 4.11: KPI per "Fattura ADSL/Fibra" as-is vs to-be

Tutto il percorso che ha visto come protagonista il cambiamento del prototipo è stato svolto in modo tale da seguire le richieste degli end-user, da avvicinarsi al loro modo odierno di lavorare, ma di apportare anche quelle migliorie che permettessero una semplificazione del loro lavoro.

Inotre come si è potuto vedere dalla raccolta dei KPI e dal loro confronto, la sostituzione con il nuovo applicativo apporta dei vantaggi non indifferenti in termini di tempistiche di gestione della chiamata. L' AHT risulta migliore del 25% circa, andando a raggiungere anche in alcuni casi un miglioramento sensibile del 35% circa.

Il numero di click effettuati dall'operatore risulta di circa 1/3 rispetto alla situazione as-is e il numero di schermate aperte si sposta da una media di una decina, ad una sola.

Infine la facilità di ricerca delle informazioni e la possibilità di reperire molte di queste direttamente sulla landing page, senza andare ad aprire alcuna schermata o effettuare alcun click, aumenta notevolmente la user experience, andando a migliorare la qualità del lavoro svolto dagli end-user.

5 | Conclusioni

5.1 Considerazioni finali

La decisione di adoperare l'approccio Agile è risultata fondamentale per la riuscita della prima fase di questo progetto. Tale scelta è stata premiata dall'approvazione da parte dell'azienda Cliente a proseguire nelle fasi successive, che prevedono la realizzazione e l'implementazione vere e proprie dell'applicativo in tutti i Call Center di appartenenza.

In un ambito come quello appena analizzato, l'utilizzo di tale metodologia è risultato dunque la scelta più efficace. Il mercato di riferimento in cui si trova l'azienda e lo scopo dell'applicativo stesso, fanno sì che siano richiesti tempi rapidi di sviluppo e implementazione dell'applicativo.

Questo progetto infatti prevede una vera e propria Transformation dei sistemi IT dell'azienda, ovvero prevede di rimpiazzare completamente il vecchio sistema e andare a cambiare l'architettura, i software, gli hardware e il modo in cui si immagazzinano e si accede ai dati.

Per questo, a causa degli impatti di questo cambiamento, le tempistiche in cui far avvenire la Transformation devono essere brevi e nello specifico sono state stimate all'incirca intorno ai dodici mesi.

La rapidità con cui è stato previsto il time to market, ha richiesto che venisse usato un approccio più snello come l'Agile, che consentisse di ridurre i tempi rispetto ad un approccio Waterfall, che avrebbe richiesto almeno il doppio del tempo per la sola raccolta dei requisiti.

Inoltre trattandosi di un applicativo che è stato scelto per migliorare la user-experience degli operatori dei Call Center, è stato necessario utilizzare una metodologia Agile, in modo tale da prevedere il coinvolgimento degli end-user, durante la fase di raccolta

dei requisiti da utilizzare per sviluppare il sistema. Senza il contributo dei Rep si avrebbe avuto una vista parziale delle funzionalità che si sarebbero dovute implementare sull'applicativo.

Questo è risultato fondamentale per riuscire a cogliere le funzionalità che l'applicativo dovrà presentare e, solamente grazie all'approccio iterativo e incrementale, che consente costantemente di adattare e reindirizzare le aspettative sul sistema in intervalli di una settimana, è stato possibile ridurre la fase di raccolta dei requisiti e sviluppo del prototipo, che altrimenti sarebbe durata quasi il doppio.

L'Agile infatti, grazie a questo meccanismo di prova e riprova, che avviene in intervalli temporali di breve durata, generalmente 2 settimane, riesce a ridurre i rischi e a soddisfare le aspettative di tutti gli stakeholder. Questo, unitamente al fatto che la fase di raccolta dei requisiti e lo sviluppo del sistema avvengano in contemporanea, permette di ridurre i costi e il time to market.

L'Agile inoltre, non solo tiene conto di tempi e costi come elementi di valutazione durante il business case inziale, ma utilizza anche un terzo elemento, ormai divenuto fondamentale per qualsiasi impresa: il business value. Questo infatti non misura tanto la redditività in termini economici, quanto il valore che il progetto può creare per gli end-user del sistema, i clienti, i manager, i fornitori ecc.

La necessità di tener conto delle esigenze di più parti interessate, fa sì che il numero di requisiti cresca fortemente. Questo può essere gestito dall'approccio Agile, in quanto si vanno a prioritizzare i requisiti raccolti, in modo da sviluppare solo quelli ritenuti più importanti.

In Accenture solitamente viene fatto uso della serie di Fibonacci per attribuire un punteggio alle user story durante la fase di raccolta dei requisiti. Questo permette di attribuire delle priorità ai requisiti e ad andare a sceglierne altri di simil valore, qualora alcuni dovessero essere sostituiti.

Nel caso specifico di questo progetto non è stato necessario utilizzare tale tecnica perché a differenza di ciò che avviene spesso durante la fase di raccolta dei requisiti, le

richieste sono state recepite attraverso un Focus Group e non tramite interviste individuali in cui ciascun utente riporta le proprie User Story su dei cartoncini.

Nel corso del Focus Group i partecipanti influenzano vicendevolmente le proprie opinioni, permettendo di indirizzare molteplici pareri e favorire un'unica soluzione rispetto alle altre. I partecipanti infatti confrontano le proprie idee, discutendo su di uno specifico argomento, e grazie all'aiuto del moderatore riescono ad arrivare ad un accordo su quale sia la scelta migliore tra quelle proposte.

In questo modo il numero dei requisiti diminuisce notevolmente e l'unico lavoro che deve essere fatto dai membri del Team è quello di raggruppare i requisiti in modo ordinato, eliminando le richieste non fattibili dal punto di vista tecnico o che sono contrarie alle direttive dell'HQ dell'azienda degli end-user.

Nel caso invece di interviste singole in cui ciascun operatore riporta i propri requisiti su dei cartoncini, il numero di richieste aumenta, spesso ripetendo la stessa esigenza in modo leggermente variato. Il Team allora deve successivamente svolgere un lavoro di confronto per riuscire a comprendere quali siano le richieste migliori o più complete, che devono essere effettivamente implementate. Questo comporta che venga utilizzato un sistema di punteggi, per far sì che il lavoro avvenga seguendo dei razionali.

In questo secondo caso data la numerosità dei requisiti e la necessità di velocità richiesta dall'approccio, spesso risulta necessario un trade-off nella fase di confronto. Questo aumenta il rischio di avere come risultato finale un'analisi poco esaustiva di tutto il contesto, che può portare a tralasciare requisiti importanti tra quelli da implementare.

Date le tempistiche estremamente ristrette di ciascuno Sprint e il numero e tipo di attività da svolgere, nell'ambito di questo progetto è stato preferito utilizzare il Focus Group, piuttosto che la tecnica che prevede la raccolta delle User Story e una loro successiva priorizzazione.

All’interno dell’approccio Waterfall invece vengono analizzati nel dettaglio tutti i requisiti raccolti in fase iniziale, senza preoccuparsi eccessivamente di prolungamenti nelle tempistiche. Infatti in un approccio di questo tipo, che solitamente richiede anni, ritardi di un paio di giorni possono essere riassorbiti nel corso del progetto. Nelle metodologie Agile invece, dove le interazioni durano in media 2 settimane, un ritardo di 2 giorni non può essere contemplato.

Per questo in progetti di grosse dimensioni, dove il time to market non è stringente ed è richiesta un’analisi approfondita di tutti i requisiti, un approccio a cascata può risultare più efficace.

Inoltre l’Agile richiede un coinvolgimento di tutte le parti interessate, dal Team al Cliente, e, vista la sua complessità, un grado di maturità di tutti i membri che lavorano allo sviluppo dell’applicativo che non sempre è presente tra le persone che lo utilizzano. Qualora venissero meno questi due elementi, risulterebbe preferibile utilizzare una metodologia più lineare, al fine di incorrere in minori rischi di fallimento, dovuti ad uno scarso commitment e ad una scarsa autonomia dei membri del Team.

Per questo motivo, nel caso si decidesse di utilizzare l’approccio Agile, risulterebbe fondamentale la scelta delle persone da coinvolgere nel lavoro. La presenza di anche solo un membro del Team che non è in grado di svolgere le attività assegnate nelle tempistiche previste, metterebbe in pericolo la riuscita del progetto.

Infine l’Agile risulta una scelta migliore quando si ha già una soluzione esistente intorno alla quale vengono raccolti ulteriori requisiti. Nel caso del progetto analizzato infatti si è partiti da una soluzione di base già esistente, la quale è stata successivamente arricchita con tutte le richieste provenienti dai Call Center.

Nel caso invece in cui andassero raccolti i requisiti per costruire un applicativo da zero, risulterebbe più efficace utilizzare la metodologia tradizionale, che esegue un’analisi critica di ciascun requisito e permette una costruzione bottom up dell’applicativo.

In seguito, per completezza informativa, vengono riassunte in Tabella 5.1 alcune situazioni in cui è preferibile un approccio piuttosto che un altro:

Waterfall	**Agile**
Nessun problema di time to market	Time to market stringente
Progetto di grandi dimensioni	Progetto snello
Requisiti calati dall'alto	Partecipazione end-user
Basso numero di requisiti	Elevato numero di requisiti
Analisi approfondita di tutti i requisiti	Possibilità di trade off tra numero di requisiti e velocità
Basso livello di maturità del team	Alto livello di maturità del team
Basso livello di commitment	Alto livello di commitment
Approccio bottom up (si parte da zero e si costruisce la soluzione)	Approccio top down (esistenza di una soluzione intorno a cui si costruiscono i requisiti)

Tabella 5.1: Waterfall vs Agile

Come si può evincere, nonostante i molti pregi dell'Agile non sempre il suo utilizzo si adatta a tutti i contesti, per questo risulta fondamentale soffermarsi ad analizzare le condizioni a contorno del progetto e tutte le variabili in campo, in modo tale da riuscire a scegliere l'approccio più corretto per sviluppare il sistema in modo efficace.

BIBLIOGRAFIA

[1] Roger S. Pressman. Software Engineering: a practitioner approach. 7th. Edition, Mc Graw Hill, 2010.

[2] Herbert D. Benington. «Production of large computer programs».In: Symposium on advanced programming methods for digital computers (1956).

[3] Royce, Winston (1970), "Managing the Development of Large Software Systems"(PDF), Proceedings of IEEE WESCON26 (August): 1–9

[4] Bell, Thomas E., and T. A. Thayer. Software requirements: Are they really a problem Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, 1976.

[5] "Waterfall Software Development Model". 5 February 2014. Retrieved 11 August 2014.

[6] McConnell, Steve (1996). Rapid Development: Taming Wild Software Schedules. Microsoft Press. ISBN 1-55615-900-5

[7] Arcisphere technologies (2012). "Tutorial: The Software Development Life Cycle (SDLC)".Retrieved 2012-11-13.

[8] Hughey, Douglas (2009). "Comparing Traditional Systems Analysis and Design with Agile Methodologies". University of Missouri – St. Louis. Retrieved 11 August 2014.

[9] Bradac, M., D. Perry, and L. Votta, “Prototyping a Process Monitoring Experiment,” IEEE Trans. Software Engineering, vol. SE-20, no. 10, October 1994, pp. 774–784

[10] Dr. B V Ramana Murthy, 2 Prof. Vuppu Padmakar 3 Ms. A. Vasavi, Department of CSE, Department of CSE Dept. of CSE, Jyotishmathi College of Technology, Chilkur Balaji Institute of Technology, JCTS, Shamirpet and Science, Shamirpet, India Aziz nagar, Hyderabad, India Hyderabad, India
Volume 4, Issue 6, June 2014 ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering Research Paper: Comparative Analysis of Software Development Process Models. IOSR Journal of Engineering (IOSRJEN) ISSN: 2250-3021 Volume 2, Issue 7(July 2012),

[11] IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 9, SEPTEMBER 2002An Introduction to Rapid System Prototyping. Fabrice Kordon and Luqi, Fellow, IEEE

[12] Hanna, Mary Farewell to waterfalls? Software Magazine May 1, 1995

[13] Nabil Mohammed Ali Munassar1 and A. Govardhan2, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010 ISSN www.IJCSI.org "A Comparison Between Five Models Of Software Engineering"

[14] Shaw C. Dibeehi Q., Walden S. Customer experience future trends & insights, Palgrave MacMillan, Houndmill Basingstoke RightNow 2011 Customer Experience Impact Report: Getting to the Heart of Consumer and Brand

[15] Manifesto Agile http://www.agilemanifesto.org/iso/it/

[16] Rivista Web MokaByte http://www2.mokabyte.it/cms/article.run?articleId=92U-SSB-4CC-OOX_7f000001_10411362_2b9269b4 Introduzione alle metodologie agili di Fabio Armani

[17] Ernesto Amato: Introduzione ad Agile http://agileinazione.it/2015/01/07/introduzione-agile/

[18] Scrum Methodology by Michael James, an Agile coach and Certified Scrum Trainer http://scrummethodology.com/

[19] Ken Schwaber e Jeff Sutherland La Guida a Scrum, Luglio 2013

[20] Metodo Agile e Scrum di Luigia Tauro https://tasurin.wordpress.com/2015/04/13/metodologie-agile-e-scrum/

[21] https://www.cprime.com/

[22] Extreme Programming:A gentle introduction http://www.extremeprogramming.org/

[23] Thomas Dudziak :XP Overview Methoden und Werkzeuge der Softwareproduktion WS 1999/2000

[24] Peter Coad, Jeff de Luca, Eric Lefebvre Java Modeling In Color With UML : Enterprise Components and Process, 1999

[25] Sadhna Goyal Chair of Applied Software Engineering Univ.-Prof. Bernd Brügge, Ph.D Technical University Munich: Major Seminar on Feature Driven Development Agile Techniques for Project Management and Software Engineering WS 2007/08 Serguei Khramtchenko, Comparing eXtreme Programming and Feature Driven Development in academic and regulated environments – Harvard University, CSCIE-275: Software Architecture and Engineering – 2004.

[26] Benjamin J. J. Voigt Supervisor: Prof. Dr. M. Glinz Advisor: Dipl.-Inf. C. Seybold Department of Information Technology University of Zurich, Zurich Dynamic SystemDevelopment Method Submitted by*:, 20 January 2004

[27] http://www.dsdm.org

[28] http://www.talkdesk.com

[29] http://www.qualitapa.gov.it/

[30] Emanuela Foglia e Anna Vanzago, Università Carlo Cattaneo: Metodologia e metodi della Ricerca qualitativa, dispensa per l'Anno Accademico 2010/2011

Ringraziamenti

I miei ringraziamenti più sentiti vanno a Roberto Mirandola e Ilaria Campana per il tempo dedicatomi e i preziosi consigli, che hanno portato al completamento di questo lavoro di tesi.

Ringrazio inoltre i miei colleghi per gli insegnamenti e la disponibilità che hanno sempre dimostrato durante il mio percorso di stage.

Infine mi sembra doveroso ringraziare la mia famiglia e i miei amici che mi hanno accompagnato e supportato durante tutto il percorso universitario, rendendo questi anni tra i più belli della mia vita.